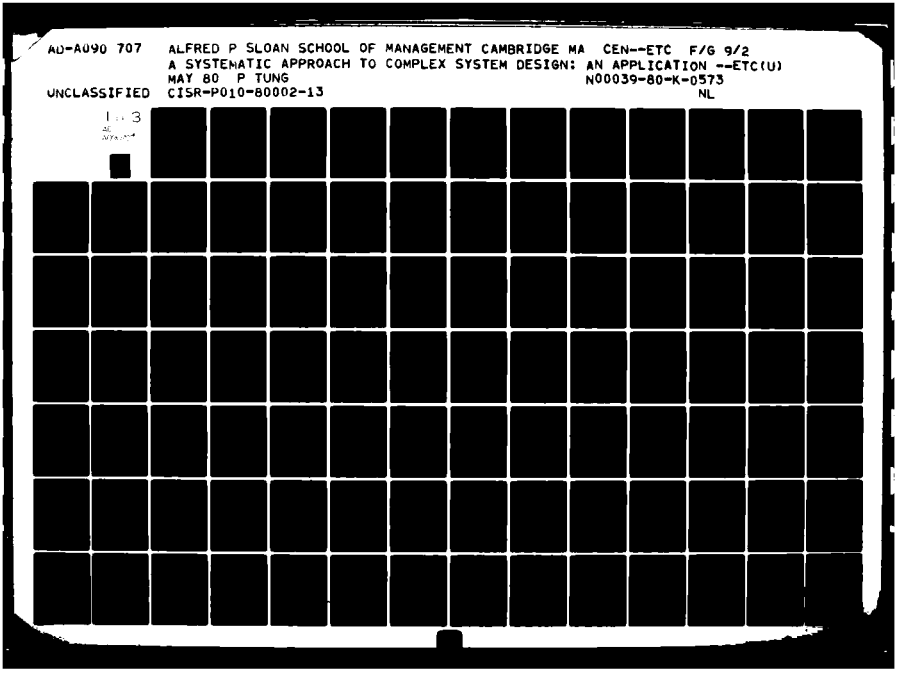
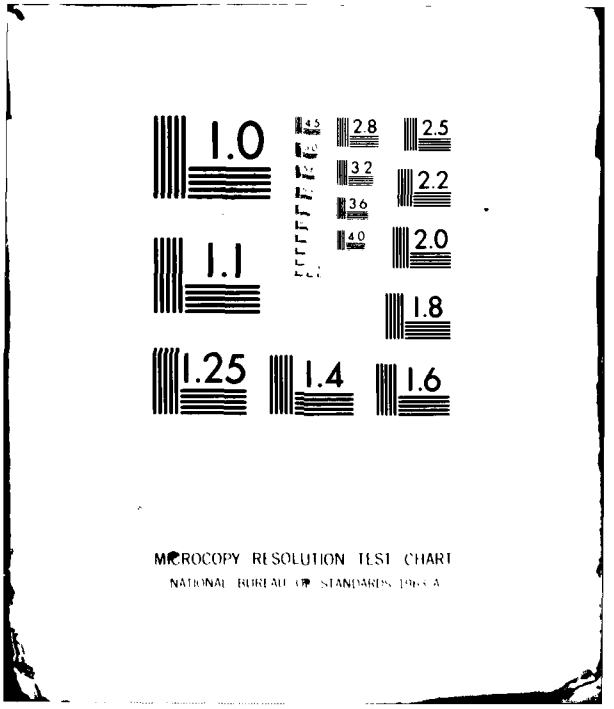
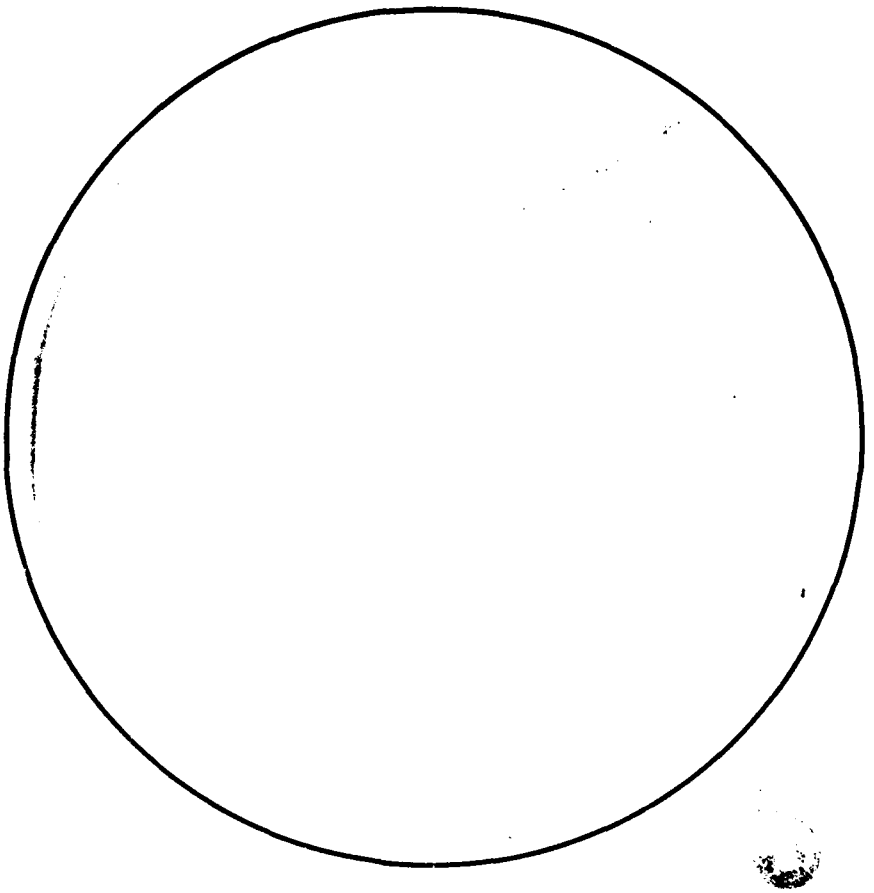
AD-A090 707 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
A SYSTEMATIC APPROACH TO COMPLEX SYSTEM DESIGN: AN APPLICATION --ETC(U)
MAY 80 P TUNG N00039-80-K-0573
UNCLASSIFIED CISR-P010-80002-13 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

LEVEL II

AD A090707



DDC FILE COPY

Center for Information Systems Research

Contract Number N00039-80-K-0573

Internal Report Number P010-80002-13

A SYSTEMATIC APPROACH TO COMPLEX SYSTEM DESIGN: AN APPLICATION TO PRINTED CIRCUIT BOARD TEST SYSTEM DESIGN

Technical Report #13

Pei-ti Tung

May 1980

A

80 10 20 002

Principal Investigator:
Professor S.E. Madnick

Prepared for:

Naval Electronic Systems Command
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: Technical Report #13
2. GOVT ACCESSION NO.: AD-A090707
3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): A Systematic Approach To Complex System Design: An Application To Printed Circuit Board Test System Design
5. TYPE OF REPORT & PERIOD COVERED:
6. PERFORMING ORG. REPORT NUMBER: P010-80002-13
7. AUTHOR(s): (10) Pei-ti Tung
8. CONTRACT OR GRANT NUMBER(s): (15) N00039-80-K-0573, N00039-78-G-0160
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Center for Information Systems Research, M.I.T. Sloan School of Management, Cambridge, Massachusetts 02139
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS:
12. REPORT DATE: (11) May 80
13. NUMBER OF PAGES: 191
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): (12) 197
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report): Approved for public release; distribution unlimited
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
18. SUPPLEMENTARY NOTES:
19. KEY WORDS (Continue on reverse side if necessary and identify by block number): Software requirements analysis; software architectural design; functional requirements specifications; problem design structuring; printed circuit board test system design
20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

The problem of designing quality software systems has existed practically as long as computers themselves. Only recently, however, have efforts been made to develop techniques to aid software designers in their jobs - in effect, attempting to add an element of science to the software design craft. One such effort

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

A

409590

The Systematic Design Methodology (SDM) is a set of concepts and techniques currently under development at MIT Center for Information Systems Research (CISR). The SDM is oriented toward assisting software designers (or design teams) in the task of structuring the architecture - the preliminary design - for a complex system.

Essential to the methodology development is the testing of the methodology. The most promising but also the most challenging testing alternative is to test the methodology against real world design problem. This thesis involves testing the methodology in a real world context where real system designers participate in applying the methodology to their design problem at hand. The application system is the test program preparation software of printed-circuit-board functional test system currently under development. Included in this report are the description of the process by which SDM was applied and an analysis of the results obtained. Insights and experiences gained in the use of the methodology are discussed throughout.

PREFACE

The Center for Information System Research (CISR) is a research center of the M.I.T. Sloan School of Management. It consists of a group of management information systems specialists, including faculty members, full-time research staff, and student research assistants. The Center's general research thrust is to devise better means for designing, implementing, and maintaining application software, information systems, and decision support systems.

Within the context of the research effort sponsored by the Naval Electronics Systems Command under contract N00039-78-G-0160, CISR has proposed to conduct basic research on a systematic approach to the early phases of complex systems design. The main goal of this work is the development of a well-defined methodology to fill the gap between system requirements specification and detailed system design.

The research being performed under this contract builds directly upon results stemming from previous research carried out under contract N00039-77-C-0255. The main results of that work include a basic scheme for modeling a set of design problem requirements, techniques for decomposing the requirements set to form a design structure, and guidelines for using the methodology developed from experience gained in testing it on a specific, realistic design problem.

The present study aims to extend and enhance the previous work, primarily through efforts in the followng areas:

(1) additional testing of both the basic methodology, and proposed extensions, through application to other realistic design problems;

(2) investigation of alternative methods for effectively coupling this methodology together with the preceding and following activities in the systems analysis and design cycle;

(3) extensions of the earlier representational scheme to allow modeling of additional design-relevant information;

(4) development of appropriate graph decomposition techniques and software support tools for testing out the proposed extensions.

This Document relates primarily to category (1) above. It reports the results of the application of the Systematic Design Methodology to the development of a design architecture for a Printed-Circuit-Board Functional Test System Software. Various techniques and methods discussed in earlier reports of this series were used in the application study. This report discusses both the development of the system's architecture per se, as well as the ways in which the methodology was used by the designers, and the lessons learned in the study.

EXECUTIVE SUMMARY

The problem of designing quality software systems has existed practically as long as computers themselves. Only recently, however, have efforts been made to develop techniques to aid software designers in their jobs - in effect, attempting to add an element of science to the software design craft. One such effort is the Systematic Design Methodology (SDM), a set of concepts and techniques currently under development at MIT Center for Information System Research (CISR). The SDM is oriented toward assisting software designers (or design teams) in the task of structuring the architecture - the preliminary design - for a complex system.

Essential to the methodology development is the testing of the methodology. The most promising but also the most challenging testing alternative is to test the methodology against real world design problem. This thesis involves testing the methodology in a real world context where real system designers participate in applying the methodology to their design problem at hand. The application system is the test program preparation software of a printed-circuit-board functional test system currently under development. Included in this report are the description of the process by which

SDM was applied and an analysis of the results obtained. Insights and experiences gained in the use of the methodology are discussed throughout.

ACKNOWLEDGEMENTS

I would like to sincerely express my appreciation to the following individuals, without whose support and encouragement this thesis would never have been possible.

I am particularly grateful to Sid Huff, who provided me with the initial motivation and with invaluable assistance in all aspects of this research.

I feel very proud and lucky to have Professor Stuart Madnick and Dr. John Howland as my thesis supervisors. Professor Madnick's effective guidance and counseling, never insufficent and always conveyed in a delightful mood, has made my thesis experience rewarding as well as surprisingly pleasant. Dr. Howland has always contributed insightful ideas, while allowing me to work out many others on my own. They both are indeed all an advisor should be and more.

I am in debt to Richie Faubert, Al Levin, and Wade Williams, for both their technical and spiritual support. Without their cooperation, this thesis would not have been carried out.

I would also like to express my thanks to Steve Hazlerig, who has been consistently and patiently proof-reading every single word of my thesis, making corrections and suggestions. I could only hope that when the time comes for him to work on his thesis, I would be able to provide him with some help.

Work reported herein was supported, in part, by the Naval Electronic Systems Command under Contract Number N00039-80-K-0573

TABLE OF CONTENTS

	Page
Executive Summary	1
Acknowledgements	3
List of Figures	7
List of Tables	7
Chapter	
1. Introduction	8
2. Software Architectural Design Activity	15
3. Background on the SDM	19
3.1 The SDM Approach to Software Architectural Design	22
3.1.1 SDM Requirements Preparation	24
3.1.2 Interdependency Assessment and Graph Modeling	27
3.1.3 Graph Decomposition	31
3.1.4 Problem Structure Interpretation	35
3.2 Development Concerning the SDM Execution	36
3.2.1 SDM Requirements Preparation	36
3.2.2 Interdependency Assessment and Graph Modeling	40
3.2.3 Graph Decomposition	43
3.3 Up-to-date Testing of SDM	45
3.3.1 Application to a Data Base Management System Design	46
3.3.2 Application to an Operating System Design	46
3.3.3 Application to the New MIT Budgeting System Design	48
3.3.4 Motivation Behind the Current Application Study	51

TABLE OF CONTENTS

4. Application System Background --
A Test Program Preparation Facility for a PCB Test System 54

4.1 System Objectives and Design Complexity Issues 58

4.2 The Simulation-Based Approach to
Test Program Preparation 62

4.2.1 Circuit Description Processing 63
4.2.2 Simulation of Input Patterns 64
4.2.3 Program Debugging Supported by Simulation 66
4.2.4 Simulation for Test Program Quality Evaluation 67
4.2.5 Simulation for Fault Isolation Preparation 68
4.2.6 Summary 70

5. Requirements Preparation of the Application System 71

5.1 Uni-functionality 75
5.2 Implementation Independence 77
5.3 Common Level of Generality 80
5.4 Assessment of the Template Approach 86
5.5 Summary 89

6. Interdependency Assessment of the Application System 90

6.1 Determining the Existence of an Interdependency 92
6.2 Weight Assessment and the Scaling Problems 99
6.3 Execution Time for Interdependency Assessment Activity 103
6.4 Summary 105

7. A Design Framework for the Current SDM Application System 107

7.1 Analysis of the Design Subproblems 109
7.2 Analysis of Subroblems Interrelationships 125
7.3 Summary 137

8. Areas for Further Research and Improvements 138

8.1 The Level of Generality Issue 138
8.2 Study of Hierarchical Structuring Principle 141
8.3 Techniques for Using the Methodology 144
8.4 Analytical Techniques for Decomposition Process 144
8.5 Linkage to the Detail Design Stage 146
8.6 Summary 148

TABLE OF CONTENTS

Reference 149

Appendix

A. Final Set of Requirements for the Test Program Preparation System as Used in SDM Analysis 152

B. Interdependency Assessment Statements for the Test Program Preparation System 175

C. Requirements Subsets Derived From the Best Decomposition 187

TABLE OF CONTENTS

List of Figures

		Page
1.1	A Model of the System Development Cycle	9
3.1	An Overview of the SDM Procedure	23
3.2	Graph Decomposition Example	34
4.1	Electronic Manufacturing Process	54
4.2	A Basic Scheme of the Simulation-Based Functional Board Testing	57
5.1	Requirement Statement Templates	74

List of Tables

		Page
7.1	Subproblem Summary Description of the Best Located Decomposition	110
7.2	Interdependencies Between Requirements Subsets in Best Decomposition	126
7.3	Statistics for Inter-Subproblem Linkages	128
7.4	summary Descriptions of the 13 Inter-Subproblem Linkages	135

1. INTRODUCTION

Not too many years ago, hardware costs were the overriding expenditure in a system development effort. Current trends point precisely in the opposite direction: the hardware costs have been decreasing as a result of advances in hardware design and manufacture. On the other hand, complex software systems are often found to be very costly, unreliable, hard to modify, and not particularly adaptable to user requirements. Such trends suggest that more attention should be paid to the software development process.

Studies concerning the complex software system problems have been conducted in the context of the system development cycle (see Figure 1). It has been found that most errors detected at the late stages (e.g., implementation falling short of requirements, functional specifications not met) have their roots in earlier stages [Thayer 1975] [Fagan 1974] [Endres 1975]. This thesis work is directly concerned with one of the early development phases called architectural design, or preliminary design.

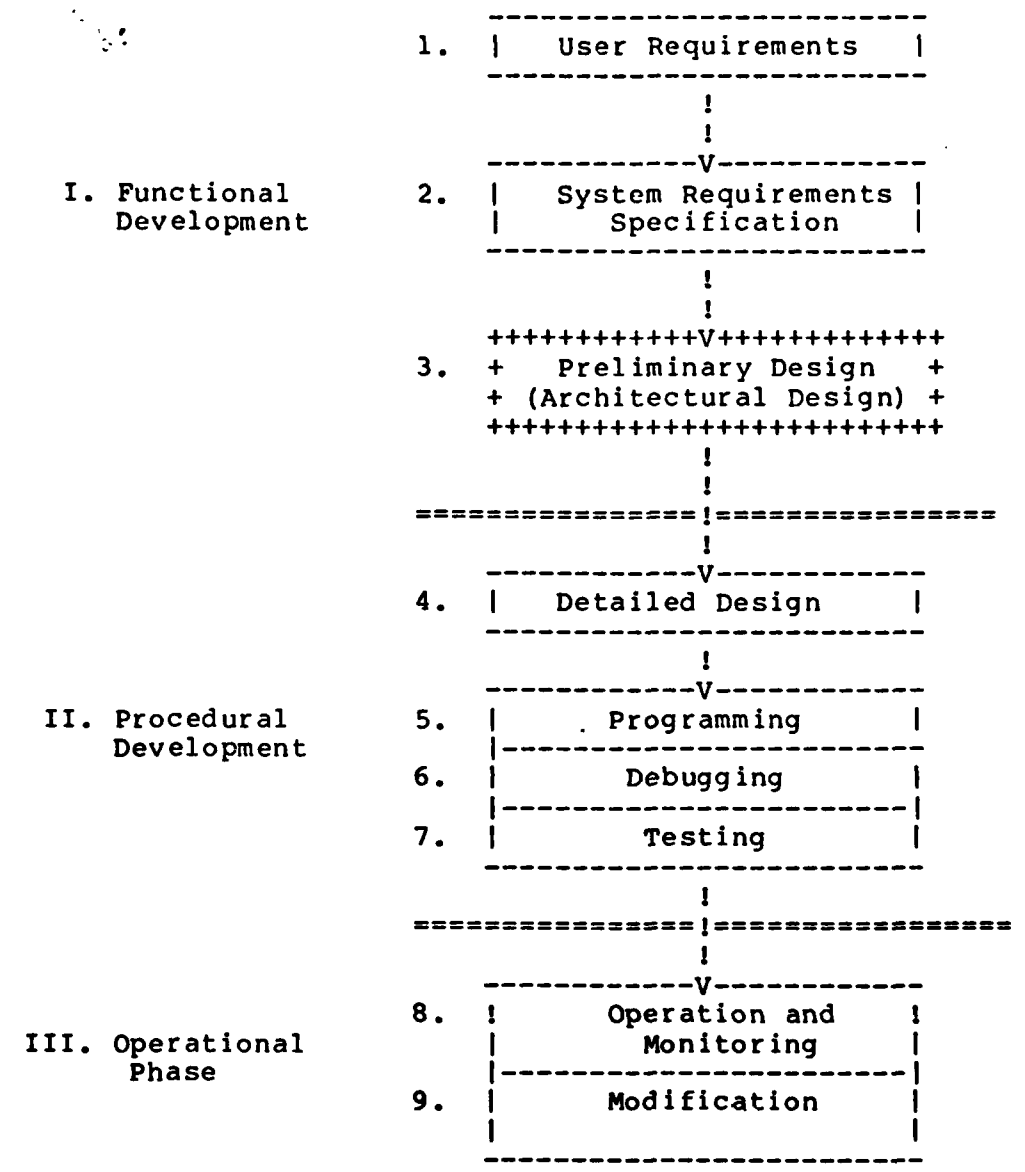


Figure 1.1

A Model of the System Development Cycle

1. INTRODUCTION

During the architectural design, the user requirements must be analyzed and a system structure identified. This activity is typically done in an unstructured ad-hoc manner, without any underlying methodology. It is very difficult for the designer(s) of a complex system to conceptualize the structural characteristics of the design problem at hand. This is because it is very hard for them to keep all relationships and trade-offs among design variables in mind simultaneously. Consequently, design decisions are rarely coordinated in a way consistent with the requirements established for the system under development. This in turn causes most of the inconveniences that typically characterize operational systems (e.g., cumbersome maintenance, cost overruns, etc.)

A few authors have only recently recognized the importance of the development of a good system architecture. For example, White and Booth are concerned about "hidden interaction" among system components resulting from overlooking design interdependencies. Often, such interactions are responsible for cumbersome system maintenance which requires major re-design efforts [White and Booth 1976]. Beldford sees the software engineer "isolated" from the original set of requirements in the sense that design implications stemming from the initial system

1. INTRODUCTION

requirements are often overlooked by organizing the eventual system in pre-defined and unjustified ways, i.e. in ineffective ways that tend to be derived more from previous experience with similar designs than from a systematic investigation of what makes best sense in the present case [Beldford et. al. 1976].

Although the software design issues have been one of the most important problems addressed by the researchers during the last three years [Wasserman et. al. 1978], it has been observed that most of the software design research effort is targeted toward the task of detailed software design (*). Some of the leading authors and researchers in the software design field recognize the importance of the architectural design task; however, they tend to view it as something that must "somehow" take place before various software design methodologies and techniques, which they have developed for the purpose of detailed design, may be applied (**).

(*) A more detailed discussion on "Software Design Research" is presented in Chapter 2.2 of Huff's Doctoral Thesis [Huff June(b) 1979].

(**) Both Myers and DeWolf have expressed this view [Myers 1978] [Dewolf 1977]. A quote from Myers' Structure/Composite Design is presented in the next chapter to illustrate this point.

1. INTRODUCTION

Although the need for the software architectural design activity was recognized by these and other authors, no real guidance as to how this activity ought to be performed was given. Having noticed this need and lack of research effort targeted toward the software architectural design, the Center for Information Systems Research (CISR) of the M.I.T. Sloan School of Management has undertaken a research project. The central focus of this research project is the development of a useful methodology - the Systematic Design Methodology, or SDM - for performing the software architectural design activity. The methodology aims at constructing a system framework consistent with system requirements. As well as methodology development, the SDM research work also includes investigation of two other related issues: testing and evaluation of the methodology. This thesis is concerned with these two issues. The bulk of the thesis work involves application of SDM to a real world complex system design problem.

* * *

The rest of the thesis is organized as follows.

1. INTRODUCTION

In Chapter 2, we clarify the problem by examining the activity that the software architectural design stage is concerned with.

In Chapter 3, we introduce the SDM approach to software architectural design in a step-by-step fashion. Then the up-to-date development concerning the execution of different methodologial steps is briefly reviewed. Finally, the up-to-date testing of the SDM is discussed.

Chapter 4 provides a background on the current SDM application system -- a test program preparation facility for a printed-circuit-board functional test system. Also addressed here are the system objectives and design complexity issues.

Chapters 5 and 6 report how the requirements preparation process and the interdependency assessment activity for the current SDM application were performed respectively. Also included in these two chapters are some assessment made with respect to the two methodological steps, discussions of difficulties encountered, and lessons learned throughout the application.

Chapter 7 presents the design framework of the current application system obtained as a result of the SDM

application. Each subproblem in the design framework and each inter-subproblems link is examined in detail.

In chapter 8, we take a retrospective view of the current application and suggest areas related to the SDM for further research and improvement.

2. SOFTWARE ARCHITECTURAL DESIGN ACTIVITY

In this chapter, we clarify the problem by examining the activity that is involved in the software architectural design stage.

It must be made clear that software architectural design is different from detailed design. The detailed design stage constitutes the actual design of program modules as opposed to system design. The software architectural design stage as defined by Freeman [Freeman 1976], is concerned with the "discovery of problem structure" in the design, i.e., the identification of major subproblems of the system and the establishment of relationships between these subproblems(*).

As mentioned before, software design issues have formed the central focus of software engineering research during the last three years. However, these recently-developed software design theories, methodologies, and techniques have been targeted, in general, toward the detailed design stage. These approaches assume that the software architecture (the preliminary partitioning of the system) either just happens, or has been performed. For example, Dr. G. Myers, primary

---

(*) Here, a "subproblem" is "a subset of system requirements", not (necessarily) a program subroutine or a collection of such subroutines.

developer of the Composite Design Methodology points out:

> "If the product being developed is a system, rather than a single program, there is another design process that must occur between the external design process and the use of composite design. This process, called system design, is the decomposition of the system into a set of individual subsystems or individual programs. Although some of the ideas of composite design are appropriate here, and some people have claimed to have used composite design for this process, composite design does not appear to be directly applicable to system design. Therefore, when designing a system, as opposed to an individual program, the designer must first partition the system into distinct subsystems or programs. Then the methodology of composite design can be used to produce the structure of these individual pieces." [Myers 1978]

The idea of partitioning a system into smaller, more manageable subproblems is not new. It has been widely advocated as a way to simplify the design of a complex

system. For example, it is very common to approach system decomposition from a strictly functional viewpoint. This may even lead to an implementation structure directly, i.e., one program module per subfunction. However, a functional system decomposition is not necessarily sufficient from a design standpoint. It is likely to miss the requirements not considered explicitly; e.g. the performance requirements are often neglected in this decomposition strategy. In other words, a functional system decomposition does not necessarily form a design problem structure.

Just what is a design problem structure? A design problem structure is concerned with how different system parts interact from a design standpoint. That is, what parts can be designed independently of others as opposed to what parts must be designed at the same time. A design problem structure is then used to identify the trade-offs that must be taken into account between competing solutions(*) to the design. Thus, while it can make perfect sense to organize several well defined system functions as separate parts in the final system implementation, it may be necessary and meaningful to organize their design in the same design

---

(*) "Competing solutions" refer to the set of implementation techniques used together to satisfy the system requirements.

subproblem if these functions are such that they need to share some system resources.

At this point, we should have a better understanding of the software architectural design activity. The need and lack of a more systematic approach to perform this activity has led to the development of SDM. In the next chapter, we provide a general background on the methodology.

3. BACKGROUND ON THE SDM

The objectives of the architectural design stage, as discussed in the proceeding chapter, can be summarized as follows. This stage should consider the interactions among requirements explicitly to identify the design structure with the following properties:

(a) its individual components are relatively independent,
(b) existing dependencies can be easily understood, and
(c) all the interactions are not hidden but explicitly made.

A more structured approach is required to the software architectural design activity in order to achieve the above objectives. The need and lack of such a structured approach led to the development of the SDM.

The SDM research was initiated by Andreu, who in the course of his doctoral thesis [Andreu 1978] formulated some key principles, and developed a first-order implementation, of the SDM approach. The Phase-I SDM was tested against two

non-trivial system design problems. One is the application to a Data Base Management System (DBMS) design performed by Andreu himself [Andreu November 1977]. The other is the application to an Operating System (OS) design performed by Holden [Holden 1978]. The two tests of the SDM were reasonably successful. Lessons learned from these tests suggested directions for further research and improvements of SDM.

Continued research work on SDM was performed by Huff, who in the course of his doctoral thesis [Huff June(b) 1979], made significant extensions to the methodology. While many of the earlier approaches have been advanced or replaced, the basic foundations laid out by Andreu were proved to be solid, and continue to endure. The Phase-II SDM has been applied to two real world design problems. One is the application to the new M.I.T. Budgeting System [Huff June(a) 1979]. The other is the application to the test program preparation software of a Printed-Circuit-Board (PCB) Functional Test System, the experience and results of which is reported in this thesis.

* * *

3. BACKGROUND ON THE SDM

An interesting as well as important point that should be mentioned here is that the underlying philosophy of the SDM is mainly based on the research work done by Alexander. Alexander, a design theorist whose main focus has been city design, developed both the philosophy and techniques of a systematic approach for the city design activity [Alexander 1964]. Upon examination of the motivations that led Alexander to devise such an approach, the initial SDM researchers found that the motivations were very close to what had moved them to look for more systematic approaches to the process of architectural design in software development. The SDM researchers were thus attempting to adopt Alexander's strategy to the software architectural design problem. How Alexander's motivations and approach were examined by the SDM researchers is reported in both Andreu's and Huff's theses (see [Section II.3, Andreu 1978] and [Section 2.1, Huff 1979]).

* * *

The rest of this chapter is organized as follows. First, we introduce the SDM approach to the software architectural design, in a step-by-step fashion. Then the up-to-date development concerning the execution of different

methodological steps is briefly reviewed. Finally, the up-to-date testing of SDM is discussed.

3.1 THE SDM APPROACH TO SOFTWARE ARCHITECTURAL DESIGN

Recall that the software architectural design activity is concerned with identifying a design problem structure, that is, the identification of major subproblems of the system and establishment of the relationship between these subproblems(*). The SDM aims at providing a more structured approach to this activity.

In this section, we briefly describe each step involved in applying the SDM. For a more complete discussion, see [Andreu 1978]. The overall SDM procedure is illustrated graphicaly in Figure 3.1.

(*) Again it must be remembered that a "subproblem" should be thought of as a "subset of system requirements," not (necessarily) a program subroutine or a collection of subroutines. To structure the software programming modules required to implement the system is a task of the detailed design rather than the architectural design.

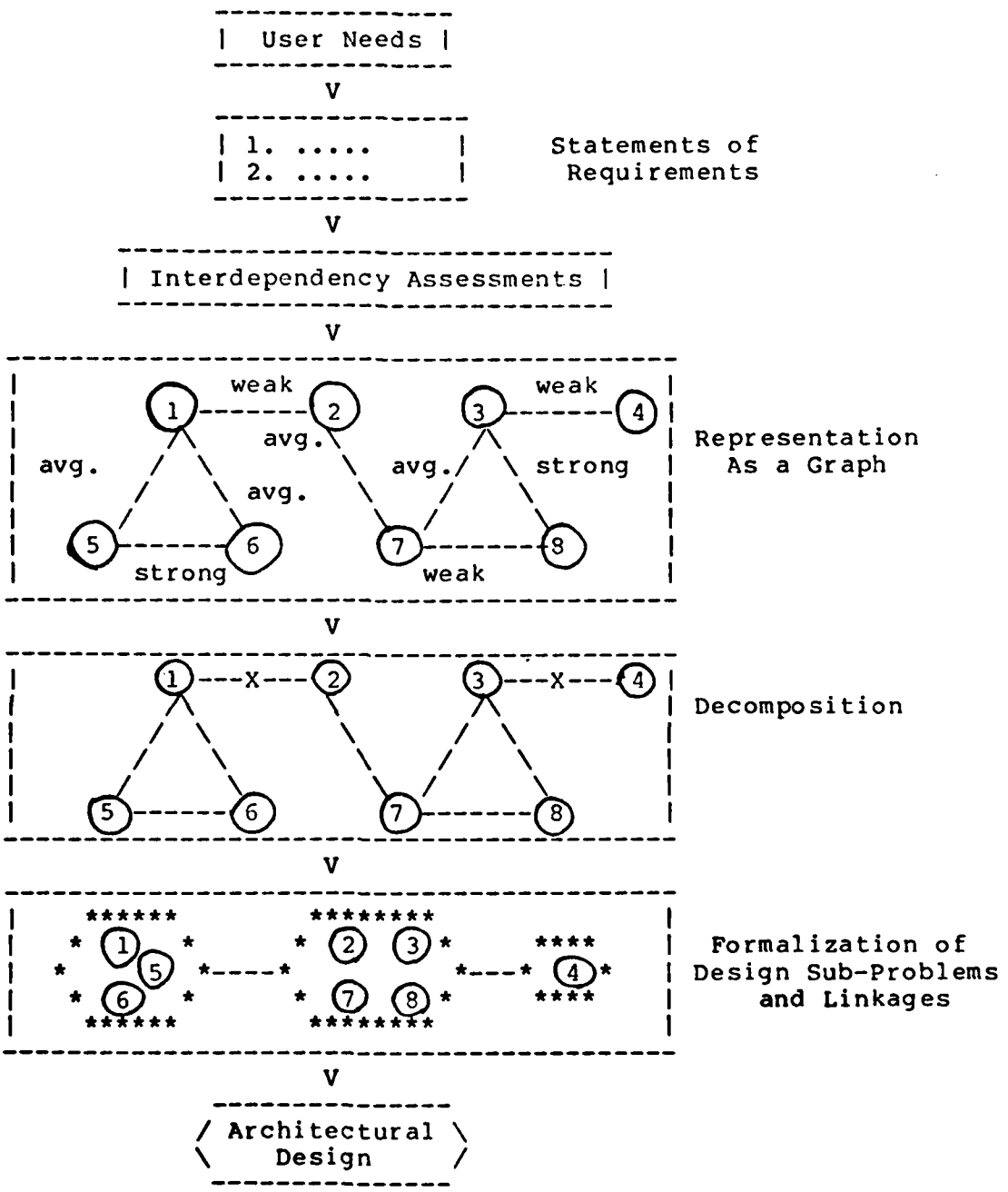


Figure 3.1
An Overview of the SDM Procedure

3. BACKGROUND ON THE SDM

3.1.1 SDM REQUIREMENTS PREPARATION

The SDM starts with an initial statement of requirements. For the purpose of SDM application, the initial statement of requirements must be expressed as a set of individual English-language statements having certain characteristics, namely:

(a) Unifunctionality:

each statement clearly and concisely specifies one specific function.

Examples of requirement statements containing this characteristic are shown below:

"New data files can be created,"
"Data files can be deleted," and
"Data files can be modified."

An example of a requirement statement that does not have this characteristic is

"Data files can be created, deleted, and modified,"
where one single statement specifies three functions.

(b) Implementation independence:

each statement specifies what is to be done but not how.

This can be illustrated by examining the following three statements:

"The Employee Salary Information File can be read by authorized users only,"

"There will be a read-access password associated with the Employee Salary Information File and this password will be given to authorized users only. No read-access to the file can be obtained without presenting the password," and

"A list of user identifications for those who have been authorized to read the Employee Salary Information File will be maintained. Any read-access to the file will be checked against this list."

All of the above three statements concern security guard against unauthorized read-access to the Employee Salary Information File; however, only the first statement specifies nothing more than the desirable function. The other two statements not only state the function, but also include the procedural information concerning how to implement the function (one uses a "read-access password" approach; the other uses a "maintaining a list of authorized user identifications" approach). The first statement is considered to be implementation independent, for it does not state any specific implementation approach. The other two are considered implementation dependent.

(c) Common level of generality:

> all statements should be, to the extent possible, at the same level of generality. Furthermore, they should be at a level high enough for both designers and users to understand.

Examples drawn from the functional requirement specifications of a computer-based budgeting system are presented below to show how requirements can be stated at different levels of generality. Consider the following two statements:

"Automate as many manual procedures as feasible to save time and effort," and

"Add a box to the Personnel Action Form to indicate whether person hired is a replacement or an addition."

There is a rather substantial difference in the level of generality between the two.

3.1.2 INTERDEPENDENCY ASSESSMENT AND GRAPH MODELING

Once the initial set of requirement statements are generated, the system designers must perform a pair-wise interdependency assessment activity. This is done by examining each pair of requirements in turn, and making a decision as to the degree of interdependence between the two. First of all, it is important to recall that we are interested in design subproblems, i.e., the identification of groups of requirements that can be considered at the same time for design purposes; thus the interdependencies among requirements should reflect this design emphasis. Accordingly, the interdependencies among requirements will be

defined so as to make explicit how different requirements interrelate from the standpoint of meeting them in the eventual design. There are two main ways in which one can think about requirements being interrelated in this manner:

(a) Supporting: The implementation of one requirement would support the implementation of the other.

(b) Conflicting: The implementation of one requirement would conflict with, or impede the implementation of the other.

The degree of interdependent relationship between two requirements can be determined by envisioning to what extent they would interact in the course of implementation.

Examples drawn from the SDM application to a Data Base Management System are used here to clarify the two types of interdependency concepts. The following pair of requirements are interdependent in a supporting sense because they call for somewhat similar functions which must performed in different circumstances in the eventual system:

> "Data base update can be performed by on-line user through query lanugage," and

3. BACKGROUND ON THE SDM

"Data base maintenance can be performed by batch."

One may consider that since in an on-line environment the usual priority is quick response time, a possible implementation to meet the first requirement is to "patch up" the performed updates while consolidating updates can be done at the same time as bulk data base maintenance, assuming that such maintenance is performed often enough. A pair of requirements that are interdependent in a conflicting sense are shown below:

"There will be transaction history facility," and

"Data integrity will be maintained with respect to active request cancellation."

In order to keep an accurate transaction history, cancelled requests should be accordingly deleted. There are several ways of implementing this, including the possibility of recording a "delete request" transaction, but the point is that the second requirement poses a "delete" capability requirement in the transaction history which is not implied by the first alone.

3. BACKGROUND ON THE SDM

Keeping the two interdependency concepts in mind, for each pair of requirements, the designer identifies interdependencies by considering several (possibly incompatible) implementation schemes. It must be made clear that alternative implementation schemes as opposed to one concrete implementation technique should be considered. The purpose here is to avoid implementation biases traditionally produced by considering only one particular implementation technique at the outset.

Designer intuition and judgement play the central role in performing the interdependency assessment. It should be noted that this activity must be somehow performed by the designer, whether using the SDM or not. The advantage of the SDM approach is that requirements may be treated a pair at a time, thus reducing the complexity of the overall task at the cost of additional analysis time.

Information about the interdependencies among requirements can be represented as a graph: the requirement statements are graph nodes and the interdependencies are weighted links (the weight of a link represents the degree of the interdependency assessed). An example of such a graph was shown in Figure 3.1.

3.1.3 GRAPH DECOMPOSITION

The next step of the methodology is to partition the graph. The partitioning is done based on cluster analysis and graph decomposition techniques. To provide an objective function for the partitioning, the SDM incorporates a quantified measure for grading the decomposed structure. This measure is based upon the concepts of module strength and inter-module coupling. Alexander [Alexander 1964], Stevens [Stevens 1974], Myers [Myers 1978], and other authors have argued convincingly that a good software design is one that consists of modules that possess high strength, or internal binding, and which simultaneously are weakly interconnected. In the SDM, this "strength/coupling" criterion is quantified in the following way. Suppose the graph representation of the target design problem has been decomposed into a set of non-overlapping subgraphs

$$\{G1 , G2 , \ldots , Gn\}$$

Then if Si = the strength of subgraph Gi, and Cij = the coupling between subgraphs Gi and Gj, we define

$$M = \sum_{i=1}^{n} Si - \sum_{j=1}^{n-1} \sum_{i=j+1}^{n} Cij$$

and use M as a figure of merit for the decomposition. The Si and Cij factors are themselves defined in terms of the number

and weight of links within a given partition, and interconnecting two partitions, respectively. Various arguments regarding how Si and Cij ought to be defined in the case of the graph model (with weighted links) are discussed by Huff [Huff February 1979], and will not be repeated here. The following definitions for these quantities were given:

$$Si = \frac{Li - (Ni - 1)}{\frac{Ni\ (Ni - 1)}{2} - (Ni - 1)} * \left(\frac{Wi}{Li}\right)$$

where

Li = the number of links contained within subgraph i,

Ni = the number of nodes contained within subgraph i,

Wi = the sum of the weights on the links in subgraph i.

$$Cij = \frac{Lij}{NiNj} * \left(\frac{Wij}{Lij}\right)$$

where

Lij = the number of links connecting nodes in subgrpah i to nodes in subgraphs j,

Ni,Nj = the number of nodes in subgraphs i, j respectively.

Wij = the sum of the weights on the links connecting nodes in subgraph i to nodes in subgraph j.

To see how these functions work in a calculation, consider Figure 3.2. Computations show that

$$L1 = 6, \quad L2 = 7, \quad L12 = 3$$

$$N1 = 5, \quad N2 = 6,$$

$$W1 = 3.2, \quad W2 = 3.8, \quad W12 = 1.2$$

As a result

$$S1 = (6-4)/(5(4)/2-4)*3.2/6 = 0.18,$$

$$S2 = (7-5)/(6(5)/2-5)*3.8/7 = 0.11,$$

and

$$C12 = 1.2/(5(6)) = 0.04 .$$

Finally,

$$M = S1 + S2 - C12 = 0.25.$$

3. BACKGROUND ON THE SDM

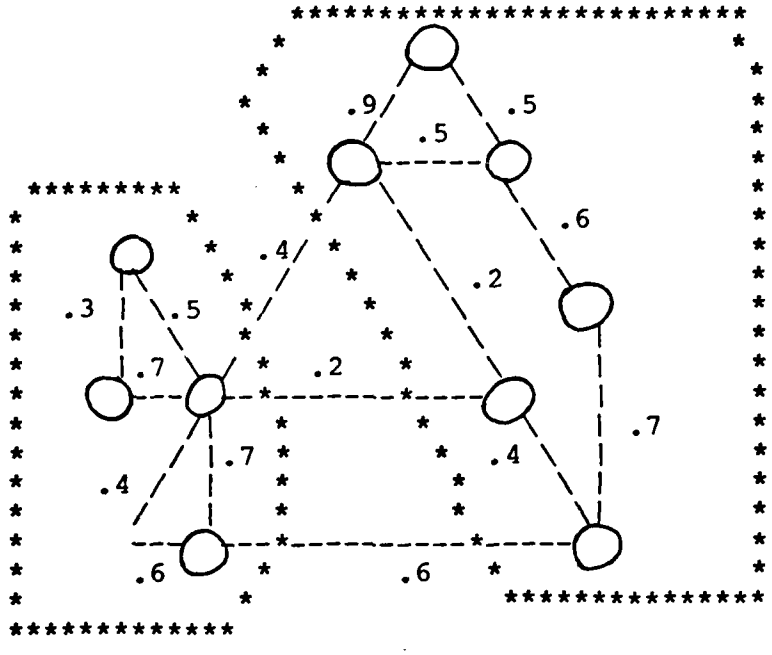


Figure 3.2

3.1.4 PROBLEM STRUCTURE INTERPRETATION

The resulting subsets of the requirements in the partition obtained above are to be interpreted by the designer as design subproblems. In essence, these design subproblems, together with their interconnections (also derived directly from the graph partition) constitute the preliminary design.

As is true with most design activities, iteration on the overall procedure has been found to be of value. A single pass through the SDM process usually produces a reasonable design, but this design may be improved considerably in terms of clarity and completeness by studying it for weaknesses. Typically, this is done by looking for subproblems with an unclear or unnecessarily complex functional interpretation, or cases of omitted or ambiguous specifications. After studying the weaknesses, the set of requirements will be modified to fill gaps and remove ambiguities.

3. BACKGROUND ON THE SDM

3.2 DEVELOPMENT CONCERNING THE SDM EXECUTION

In this section, we present a brief overview of the up-to-date SDM development concerning execution of each of the methodological steps. Guidelines and techniques that have been investigated, proposed, and/or implemented to help organize the activities involved in, or to realize the execution of, these methodological steps will be identified. Pointers to detailed discussion concerning each of the specific will be given.

3.2.1 SDM REQUIREMENTS PREPARATION

The SDM is driven by a set of requirements containing certain kind of characteristics.

In testing the methodology (Phase-I SDM) against the DBMS design problem, Andreu identified a list of characteristics which the set of requirements used by the SDM should possess, including

(1) implementation independence,

(2) system structure independence,

(3) independence among requirements,

(4) simplicity,

(5) no "stand alone" requirements,

(6) plausibility.

Andreu explained each of the above characteristics along with illustrations drawn from the DBMS requirements set, and justified the need for the inclusion of these characteristics based on the goal of the SDM [Andreu November 1977]. A lesson learned from the SDM application to the DBMS indicated that for the purpose of proper set decomposition, the following characteristic should be added to the above list [Andreu December 1977]:

(7) common level of generality

This list serves as the basic "check list" for SDM requirements preparation.

During the Phase-II SDM development, the transition from the "functional requirements specification" stage to the "software architectural design" stage was investigated [Huff June 1978]. Specifically, the need to capture user-level functional requirements in a form appropriate for follow-on SDM analysis (interdependency assessment, etc.) was examined. The SDM researchers' first thought in this area was that one

of the well-documented "requirement statement languages" (RSL's), which have emerged over the past few years, might proved suitable, perhaps with some modifications, to their needs. Upon examination of some of these languages and assessment of their nature and functioning with respect to SDM, the SDM researchers were led to make some general observations regarding ambiguous terminology that has grown up around these RSL's, around system requirements specification in general, and regarding the appropriate role of RSL's in the system development cycle.

To provide some clarification for the ambiguities and mis-uses that are frequently encountered in the literature in the "system requirements specification" area, the SDM researchers first examined three important aspects of requirements:

(1) degree of procedurality,

(2) level of abstraction(*), and

(3) capability-versus-process.

(*) In this thesis, the terms "level of abstraction" and "level of generality" are used interchangably.

Then a simple framework, in which these three characteristics can be viewed together, was put forth for conceptualizing and describing requirements in the context of the system development life cycle. For a detailed discussion concerning the above, see [Chapters 2 and 3, Huff June 1978]. This author has found the clarification very helpful in preparing the requirements set for the current SDM application reported in this thesis.

Having observed that the RSL's were useful tools for documentation rather than for design(*), the SDM researchers inferred that these RSL's were not appropriate for expressing SDM requirements. Instead, a new approach was proposed by Huff. The new approach is called "the template technique", as it is based on a set of seven basic "requirement statements templates" in which each template corresponds to a general category of statement type. The template approach has been used in the two Phase-II SDM testing cases and has been found helpful for SDM requirements construction. The effectiveness of the template technique in preparing the SDM requirements for the current application reported in this thesis will be assessed later.

(*) Experience of the RSL's has seen them used primarily as documentation techniques rather than design techniques.

3.2.2. INTERDEPENDENCY ASSESSMENT AND GRAPH MODELING

Once the initial set of requirements are generated, a pair-wise interdependency assessment activity can be performed. In his initial research on software architecture, Andreu employed a simple graph model to represent the functional requirements of a system and their implementation interdependencies. Each requirement is represented as a separate node; a link connecting two nodes corresponds to the existence of an interdependency between them.

As mentioned before, the interaction of a given pair of requirements can occur in two ways: supporting or conflicting (see Section 3.1.2)..Conceptual models in which implementation of one requirement is related to implementation of the other in either of the above two ways can be imagined to identify the interdependency. Andreu has proposed guidelines for the generation of conceptual models as well as procedural guidelines for performing the interdependency assessment activity. A summary of the guidelines can be found in [Holden 1978]. A detailed explanation for each of the guidelines is given in [Chapter 3, Andreu November 1977] along with examples drawn from Andreu's SDM application to the DBMS. While this basic model proved satisfactory for the early exploratory studies [Andreu

1978] [Holden 1978], it was also clear that improvements and extensions could be made so as to allow a designer to represent additional design-relevant information.

Huff, in his follow-up research on SDM, identified and analyzed various possible types of additional information that software designers would draw upon (usually intuitively) in constructing a practical architectural design. The types of the information believed to be most relevant and accessable via designer judgement and knowledge include

(1) interdependency strength,
(2) interdependency similarity relationships and accompanying strength factors,
(3) implication relationships between requirements and between interdependencies,
(4) hierarchical implication relationships.

Useful schemes needed to effectively represent these additional kinds of information in the graph model have also been developed.

In order to illustrate the application of the extended design model, a subset of 22 DBMS requirements were analyzed by Huff [Chapter 4, Huff July 1978]. A detailed discussion

of all of the proposed extensions and the experience gained in the assessments of these additional kinds of information (interdependency weights, interdependency similarities and associated weights, and implication relationships) can be found in [Huff and Madnick July 1978].

Analysis study on which extensions should be adopted for the purpose of extending the SDM, with respect to design-relevance and producing effective decomposition, was carried out. It was argued that the most significant such extension was the inclusion of a weight factor to correspond to each assessed interdependency, with a weight on each arc representing the strength for the interdependency. Thus it was decided to incorporate this "interdependency strength" extension into the representational model. Huff proposed a variety of possible ways in which such a weight could be defined and justified. The weight factor can be determined based on "how closely two requirements are related" and/or "how certain the designers are about the existence of the interaction between the two requirements." Like determining the existence of a link itself, determining the weight of a link will be made judgementally rather than mechanically.

3.2.3 GRAPH DECOMPOSITION

Once the design-relevant information pertaining to the target system is modeled as a graph, it is ready to be decomposed. The set of analysis techniques for use in the graph decomposition is central to the actual execution of the SDM. From the SDM viewpoint, the graph decomposition is basically a mechanical task. The important analytical techniques for applying the SDM concepts are reasonably well developed at this point.

The graph decomposition problem is very much dependent on the context of the graph model. Although there are certain common principles, implementation details are generally context specific. Andreu, drawing on the common principles, formulated and implemented three main techniques, i.e., similarity clustering, "leader group" clustering, and iterative partitioning, for solving the decomposition problem of the basic graph model and basic goodness measure [Chapter IV, Andreu 1978].

The decomposition problem was investigated further by Huff during Phase-II SDM development. Recall that several extensions to the basic graph model were proposed (see Section 3.2.2). Having extended the SDM representational

framework, it becomes necessary to modify the various analysis techniques so as to incorporate the information included in the new representation. The results of the phase-II SDM graph decomposition problem investigation are reported in [Huff February 1979]. Several different but related topics are addressed there, namely, factoring the interdependency weight assessment values into the decomposition analysis activities; techniques for including interdependency similarity information; new hierarchical clustering algorithms for effecting a graph decomposition; comparative analysis among the old and the new clustering methods. Huff also developed and tested a new top-down hierarchical partitioning algorithm (called "interchange partitioning algorithm") which is well suited to decomposing the particulr type of graph being dealt with in the SDM context. This new algorithm is presented in detail in [Huff March 1979].

3.3 UP-TO-DATE TESTING OF SDM

The SDM research to date has involved both methodology development and application studies. There have been four completed applications of the SDM to non-trivial design problems, including the one reported in this thesis. The Phase-I SDM was applied to a Data Base Management System [Andreu November 1977] and a small Operating System [Holden 1978]. In both cases, the studies were carried out by SDM researchers. The Phase-II SDM has been applied to two real world systems. One is an application to the new M.I.T. Budgeting System [Huff June(a) 1979], the other is an application to a major part of a Printed-Circuit-Board Functional Test System reported in this thesis. While the role of the system designers were "simulated" by the SDM researchers in Phase-I SDM applications, real world system designers' participation have been heavily involved in the Phase-II SDM applicaitions in the latter two cases.

In this section, we briefly discuss each of the applications.

3.3.1 APPLICATION TO A DATA BASE MANAGEMENT SYSTEM DESIGN

In the case of the design of a DBMS, the first application of SDM to a design problem, Andreu started with a set of requirement statements derived from a specification issued by a U.S. government agency. As a result of the application, a design framework for the problem analyzed was identified and discussed; its study pointed out that the methodology had produced interesting and unforeseen results, mainly related to the completeness of the original requirements set.

In summary, the application provided the SDM researchers with valuable insights into the potential of the explored methodology.

3.3.2 APPLICATION TO AN OPERATING SYSTEM DESIGN

In a separate application test, Holden applied the SDM to the design of a small software operating system. This test differed from the DBMS example in that the target system already existed (as a pedagogical case study in the textbook Operating Systems by Madnick and Donovan).

3. BACKGROUND ON THE SDM

Requirement specification statements were developed from published descriptions of the purpose and approach underlying the operating system. A number of iterations were required to build a reasonably clear, consistent, and complete requirements set. It was somewhat surprising that, although the target system was already built and documented, the requirement definition task was found to be the most challenging and time-consuming aspect of the test. Interrelationships between requirements were developed according to the guidelines specified in the earlier research efforts, i.e., the application of SDM to DBMS design by Andreu.

The design produced by SDM resembled the original design in most respects. Given the manner in which the requirements were generated, although perhaps not surprising, it was encouraging to see that at least in this case the SDM appeared to be stable. Holden analyzed the few interesting differences between the original and SDM designs. While it was impossible to prove the case conclusively, Holden felt that, in most cases where design differences arose, SDM had produced an alternative that appeared to be as good as, or possibly better than, the original design.

The SDM researchers believe it was fair to conclude that the two tests of the SDM had been reasonably successful. Much was learned on the basis of these tests to suggest directions for further research and improvements of SDM. (see the sections on "Areas for Further Research" in both [Andreu 1978] and [Holden 1978]).

Additional methodology development was carried out by Huff. Significant extensions were made to the Phase-I SDM. The Phase-II SDM has been tested against the following two real world applications.

3.3.3 APPLICATION TO THE NEW MIT BUDGETING SYSTEM DESIGN

The third application of the SDM is an applicaton to a medium-sized realistic system architectural design problem. The system under design is a new MIT Institute-wide, computer-based budgeting and planning system. As the earlier SDM applications had been concerned with system software - a DBMS and an OS - this application, which is a fairly conventional yet reasonably complex data processing application system, promised to provide new insights as to SDM applicability to such "application system" design.

More importantly, this application is the first one in which the key SDM design data was obtained from the system designers themselves, thus providing the first significant unbiased evaluation of the usefulness and effectivenss of the methodology. Over the course of ten meetings, Huff and the system architects for the new Budgeting System worked out the functional requirement statements and requirement interdependencies in detail. The decomposition of the resulting requirements graph and the interpretation of the design problem structure were then carried out. The final architecture for the target system was well received by the Budgeting System architects, and they have expressed their intention to use this architecture in guiding their coming detailed design work.

Throughout the application, it was found possible to execute the various steps of SDM with little difficulty. While a substantial amount of time was spent in preparing the requirements set and the interdependency assessments, the decomposition analysis and architectural interpretation were relatively straightforward and not particularly time consuming. This suggests that the time and effort invested early in the SDM analysis process pays off in terms of a "good" initial decomposition and easily interpretable architecture later on. Such an observation is in general

agreement with what other software design researchers have found in other contexts [Boehm 1973].

The Budgeting System designers have expressed both positive and negative reactions toward the SDM analysis exercise. The major negative reactions concerned with the time required for the analysis and some doubt about the overall value of the exercise. The latter occurred mostly at the outset of the analysis process. Fortunately, both issues were tempered by the designers' appreciation of the research nature of the study. The positive reactions concerned new design ideas as well as clarification and improvement of current ideas that emerged during the exercise, discovery of new ways of approaching the design task in general, e.g. separation of functional concerns from implementation issues, and their belief that the final architecture would be of assistance in the later detailed design efforts.

As pointed out by Huff, the eventual value of the resulting Budgeting System architecture cannot be known at this time. Rather, it will be necessary for the SDM researchers to follow up this exercise in the future to learn what kind of impact this application study might disseminate. For detailed report on this application study, see [Huff June(a) 1979].

3.3.4 MOTIVATION BEHIND THE CURRENT APPLICATION STUDY

The other real world SDM application -- an application to the test program preparation part of a Printed-Circuit-Board (PCB) Functional Test System -- has been performed by this author. How the methodology was applied, the results obtained, the lessons learned, and the assessments made on the methodology are reported in detail in the following chapters. In this subsection, we review the motivations behind the current application study. One motivation originates from the importance and difficulties of the SDM testing. The other motivation originates from the PCB Functionl Test System designers' interest in searching for *effective approaches to software development and maintenance.*

3.3.4.1 Difficulties of SDM Testing

An important part of methodology development is testing against application situations. In the case of the SDM, testing presents particularly difficulties.

The main difficulty lies in the fact that the SDM is specifically oriented toward large-scale, complex systems which require many man-years of effort to design, build and install. The magnitude of time and effort required to effect

a realistic test is large. Even if the size were not a problem, the detailed knowledge required concerning the specific application area necessary to fully comprehend the requirements of the system and their possible implementation alternatives presents a second difficulty.

Huff, in his Doctoral Thesis Proposal, has proposed three ways to proceed with SDM testing [Huff Oct 1978]. It is the real-world test case that is the most difficult to pursue:

"...A third alternative is the real-world test. This approach would require an agreement with some organization currently facing a medium-large-scale design and development task. ..... This kind of test would appear to be most promising in terms of identifying the actual strengths and weaknesses of the SDM, but also has the most associated difficulties, including the necessity of locating an organization willing to participate and risk the expenditure of some resources on such a test, the difficulty of adequately monitoring and controlling the test, etc..

....We believe that the likelihood of locating a suitable organizational test situation is somewhat low, but since the potential research payoff is high, we shall continue pursue the alternative...."

3.3.4.2 ABC's Participation in SDM Testing

An electronic corporation, that we will refer to as ABC, provided the environment for testing SDM (*). Having been concerned with the cost of software, ABC's software managers and designers have been searching for effective approaches to software development and maintenance. When introduced to the SDM approach, they became interested in participating in SDM testing. From the long term viewpoint, they regard the SDM application a starting experience of a more structured approach to the functional development of software system development. It was decided that the SDM be applied to a major part -- the test program preparation part -- of a PCB Functional Test System(**), a system currently under development. Before presenting the detail of the application study, in the next chapter we first provide a general background on the application system.

(*) The corporation has requested anonymity since this study involves products still under development.

(**) Since SDM is still at the testing stage and the organization is unfamiliar with it, applying it to the entire system architecture design may be a too large task to control.

4. APPLICATION SYSTEM BACKGROUND

The application system under discussion here is the Test Program Preparation part of a Printed-Circuit-Board (PCB) functional test system. PCB test is one stage of the entire electronic manufacturing process. The process, which is illustrated in Figure 4.1, consists of a series of assembly and test stages that culminates in the shipment of product.

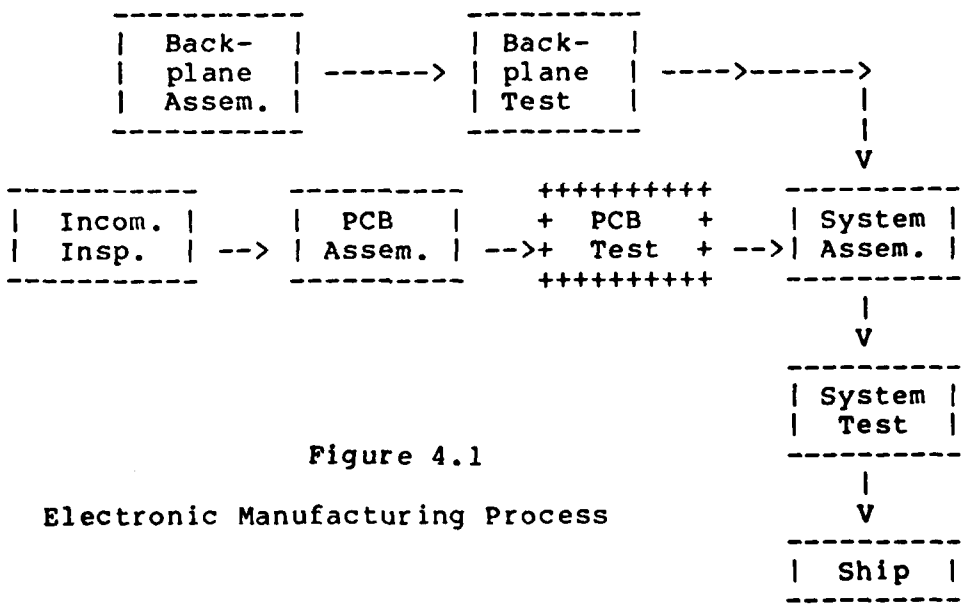


Figure 4.1

Electronic Manufacturing Process

4. APPLICATION SYSTEM BACKGROUND

At incoming inspection, individual components and bare boards are tested. Then the components are loaded onto printed-circuit boards at the board assembly stage. This is followed by a board test. In parallel, the interconnect devices -- cables, harnessess, and backplanes -- are assembled, tested, and then combined with the loaded and tested printed-circuit boards at system assembly. A system test is performed and the product is ready for shipment. Our attention is at the PCB board test stage. In particular, we are interested in the functional testing of logic-circuit boards.

Logic circuits consist of discrete semiconductor devices and integrated circuits (IC's) of small-scale, medium-scale, large-scale, and very-large-scale (SSI, MSI, LSI, and VLSI). Background information about IC characteristics and logic-circuit design is available from several sources listed in the reference [Morris 1971] [Su 1974] and is not provided here. The logic circuits may be assembled by soldering these IC's onto boards with printed-circuit interconnections, or other similar interconnection techniques. The same boards may include some analog circuits, usually as interfaces between the digital circuits and external signals. A functional board test is a test for the board's proper functioning by attempting to duplicate the final system

environment in which the board will operate. The board is typically accessed by an edge connector.

In recent years, simulation has been used as an indispensible tool for effective PCB testing [Breuer 1976] [Szygenda 1975] [Anderson 1975]. The current SDM application system is a simulation-based test program preparation facility that supports all the preparation and processing work required prior to the testing and diagnosis of the circuit boards.

A digital simulator is a computer program that models the operation of a logic circuit. The basic scheme of the simulator approach is displayed in Figure 4.2. Initially, the test engineer inputs both a description of the board and a tentative test program (input patterns to exercise the board) to the simulator. The simulator relies upon a library of models for all component types on the board. Briefly speaking, what the simulator has to do is to provide feedback to the test engineer for the development and debugging of an effective test program, and to generate the data base needed for later diagnosis.

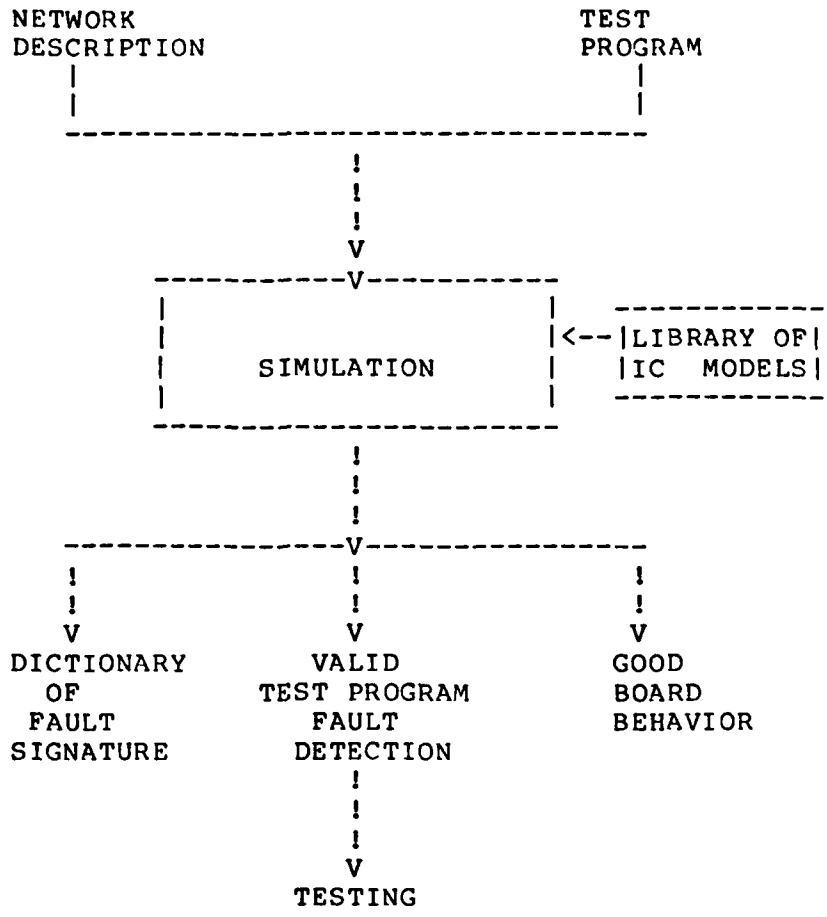


Figure 4.2

A BASIC SCHEME OF THE SIMULATION-BASED FUNCTIONAL BOARD TESTING

4. APPLICATION SYSTEM BACKGROUND

In the rest of this chapter, the motivation behind the development of the current applicaiton system and the design complexity issues resulting from the specific objectives of the system will be discussed first. Then each specific task involved in the simulation approach to test program preparation will be examined.

4.1 SYSTEM OBJECTIVES AND DESIGN COMPLEXITY ISSUES

The architects of the current application system have had years of experience in the field of PCB testing. Many of them have participated in the development (design as well as implementation) of ABC's existing test systems. The main function of the existing system is to test SSI/MSI based circuit boards. Technological changes in digital PCB designs(*) have eroded the effectiveness of the program preparation and diagnostic capabilities of the existing test system. As a result, programming times are increasing and production throughput is decreasing. The major motivation behind the development of a new test system is to solve many of the current technological problems straining the capabilities of the existing test system.

(*) Namely, the emerging of the LSI/VLSI based circuit boards.

4. APPLICATION SYSTEM BACKGROUND

The following objectives of the new test program preparation system contribute to the complexity of the system design:

(1) The new system has to support LSI/VLSI board testing at least as well as the existing test system does for the simple boards. The complexity of the LSI functions and the size of LSI devices are an order of magnitude above those of simpler boards(*). The new system must be able to properly handle the size and type of LSI devices. Some LSI devices are difficult to model. Improved IC modeling capabilities must be provided to solve this problem. Furthermore, many size boundaries which exist in the present product must be removed in order to accomodate the size of the LSI devices. Thus the new system is expected to be much more complicated than the existing one. Here we give a rough feeling about the complexity of the current system:

(a) about 70 man-years of effort were involved in the development of the current system (both design and coding);

(*) In terms of equivalent number of gates, the sizes of LSI, MSI, and SSI are about >=100, >=12, and =1 respectively.

(b) the entire system contains about 500,000 lines of assembly language code; the program preparation part alone contains about 100,000 lines of code.

The total amount of effort required for the development of the new system is expected to be at least twice as much.

(2) The new system actually aims at giving better support to board testing. That is, the new system will provide more powerful diagnostic tools. The test program preparation facility in turn must be capable of producing the necessary information to support the diagnosis. The requirement for these enhanced capabilities, such as better timing analysis, better awareness of tester pin electronic properties, etc., thus increases the complexity of the program preparation software facility. Another objective is to reduce the program preparation processing time as much as possible. In addition to using a faster processor, other techniques must be employed to achieve this goal. This objective again complicates the design. The more complicated issues arise from the fact that many trade-offs between the implementations supporting one objective and those supporting

the other must be carefully considered.

The discussion of the design complexity issues would have ended here if all we had wanted was a superior new test system that can support board testing for both SSI/MSI boards and LSI/VLSI boards. However, this is not the case:

(3) From the current users' viewpoint, the new test system must be compatible with ABC's existing test systems. The compatibility issue complicates the design in the following sense:

(a) The designer is often forced to consider more in order to ensure that compatibility can be satisfied.

(b) Sometimes the designer may think of a potentially better design; however, he may have to forego it if the new design makes some compatibility requirements either hard or impossible to satisfy.

(4) Another major objective of the new test system software is to add capabilities to the existing testers which fail to properly handle the LSI/VLSI board testing. As for the program preparation facility, it must support program preparation for boards that are to be tested on the current testers. Information generated by the new program preparation software will be sent to the current testers for testing/diagnostic purposes whenever requested by the individual tester. This objective demands "internal processing" compatibility. For example, the new internal representation of the data base should be, to the extent possible, compatible with the old internal representation of the data base. Such a demand complicates the design in that the architecture of the computer on which the new program preparation software will be run is very different from the architecture of the computer on which the current test system is run.

4.2 THE SIMULATION-BASED APPROACH TO TEST PROGRAM PREPARATION

In this section, we examine each specific task involved in the simulation-based test program preparation.

4.2.1 CIRCUIT DESCRIPTION PROCESSING

The simulator must use a logical image of the board (called a "circuit description" or "network description") in order to model the operation of the logic circuit board. This software image is provided by the programmer in the form of statements or equations which describe the circuit. The description must include the types of the logic elements on the board and the interconnections between them. These elements may be the basic gates of the logic circuit or they may be complete integrated circuits of the SSI, MSI, LSI, or VLSI category. Furthermore, a description of the interconnections between the board under test and the tester itself must also be provided.

Several simulators model the operation of a logic circuit at the gate level. The gate level approach offers the potential of greater timing accuracy; however, this approach is less efficient and may be impractical for many MSI and LSI IC's. Other simulators use subroutines to describe the functions of an IC. The functional approach results in a faster simulation that requires less memory. Models of the widely used types of logic elements are usually provided by the system in an IC library. If the logic circuit to be simulated contains an IC that is not in the

library, it is necessary to add a description of that IC to the library. Depending upon the simulator, a new subroutine may be programmed for the IC model. Alternatively, the IC may be described in the same manner in which the logic-circuit board is modeled, using other IC's already in the library.

The circuit description is then processed by a preprocessor program which checks the circuit description for correctness and consistency between the information provided and the information previously stored about the logic elements. Typical inconsistencies might include unused elements within integrated circuits, unused inputs on logic elements, and overloaded pins. The information is converted to the format required by the simulator. It is also available to the programmer in several other formats.

4.2.2 SIMULATION OF INPUT PATTERNS

The second type of data that the test programmer must provide is a set of input patterns. Manual analysis is required to select these patterns, but a more cursory analysis is possible because the simulator will determine the effects of the patterns.

4. APPLICATION SYSTEM BACKGROUND

The simulator applies each input pattern to its model of logic circuit. Some simulators also model the timing sequence in which the test system will apply these patterns to the real board. This feature ensures that the responses of the modeled circuit will more closely match those of the board.

The test programmer uses the simulator warnings and logic state information to determine what input patterns should be added or changed. He selects patterns that will eliminate indeterminate conditions as early in the test program as possible.

The logic states at every node for each test in the program can be displayed to the user. Such a display is called a "nodal status listing." It simplifies the selection of input patterns because the programmer need not remember all of the logic states created by his input patterns. He can scan the nodal status listing to locate the test steps for which the logic states at the appropriate nodes are similar to what he wants. He then can add or modify a small number of input stimuli to obtain the desired result.

The software simulator calculates the effects of the input patterns throughout the logic circuit and at the board output pins. The output test patterns thus are obtained

automatically from the simulator.

Most simulation software packages include postprocessors to convert these output patterns into the test language of the system that will be used to test the boards. The final test program will contain both input and output patterns, just as if they had been manually programmed.

4.2.3 PROGRAM DEBUGGING SUPPORTED BY SIMULATION

Test program debugging consists of determining why the actual test results differ from the expected results and modifying the test program to eliminate this difference.

Use of simulation to obtain output responses results in the easiest debugging procedures. The simulated output responses will have identified indeterminate conditions so that they can be ignored or eliminated prior to the debugging procedure. The physical board used for debugging needs not be manually tested as extensively because it does not have to be a "known-good" board. The simulation data aids the programmer in resolving the cause of any discrepancies between the expected and actual test results. If a discrepancy is due to a fault on the board rather than a program bug, the simulator diagnostics will isolate it rapidly.

4.2.4 SIMULATION FOR TEST PROGRAM QUALITY EVALUATION

The quality of a test program refers to the percentage of faults detectable by the program. A digital fault simulation program models the operation of the logic circuit board with a fault inserted, and compares the output responses with those of the modeled good circuit. If the output responses are identical, the fault is undetected and the simulator records it for later printout in an "undetected fault list." If the output responses differ, the fault is detected and the resulting output signature is recorded for fault isolation purposes (more about this issue will be discussed in the next section). The user of a fault simulator usually specifies a restricted fault set for his initial test program simulation. This saves simulation time. The fault simulator can model a much wider range of faults than can be physically inserted, including timing faults and internal IC faults.

4.2.5 SIMULATION FOR FAULT ISOLATION PREPARATION

The programming tasks described in Section 4.2.1 through 4.2.4 are sufficient to generate a comprehensive test program for GO/NO-GO testing(*). However, most of the time and cost in manufacturing and field service testing is spent isolating faults on defective board, not sorting good ones from bad. There are two basic categories of fault-isolation techniques for logic-circuit boards.

The probable cause of the failure can be predicted based upon the actual responses at the output pins of the board when the test fails. These probable causes are listed in a "fault dictionary." The digital fault simulation used to evaluate test program quality can automatically generate fault dictionaries by storing the failure signature resulting from each modeled fault. A failure signature is a set of logical values that are observed at the board output pins in the presence of a particular fault at the failing test step. These dictionaries can be stored within the test system to provide automatic fault diagnosis or they can be printed in hard-copy for manual lookup. Fault dictionaries provide a

---

(*) GO/NO-GO test is a test intended to produce a pass/fail result of board testing as opposed to provide dignostic information.

very rapid fault diagnosis, but several faults may be indicated as the probable cause of the same failure. This occurs whenever a set of faults have the identical failing signature at the failing test step. One approach to improving the diagnostic resolution is the use of additional test points within the board. Another approach is further simulation of the several probable faults so as to obtain different fault signatures.

The other techniques require probing a path backward from the incorrect output pin to the fault location. The operator probes the output node of the IC that is connected to the incorrect board output pin. If the response is incorrect, he next probes the inputs of that IC. If one of the inputs has an incorrect response, he next probes the output of the IC to which it is connected. He continues probing each node in the path until he reaches an IC that has the correct inputs and an incorrect output. He thus has located the fault as either a defective IC or a physical defect on the board at that IC output node. A guided-probe software package basically automates the analysis procedure that a technician follows during probing. It requires a data base consisting of the circuit interconnections and the correct logic states at every node. Simulation of a good logic circuit requires a network description and provides the

nodal-status information, both of which can be used for guided probing. Alternatively, the nodal-status tables may be printed in hard-copy form for use by a technician. In either case, no additional setup effort is required.

4.2.6 SUMMARY

There are both advantages and disadvantages of the simulation approach to PCB functional testing. While some of these have been mentioned in the above discussion, it is not our intention to address this issue further. At this point, the reader should have an understanding of the SDM application system currently under discussion. In the chapters that follow, we are to report how the current SDM application was performed, discuss the lessons learned, and assess the methodology based on the experience gained from the application.

5. REQUIREMENTS PREPARATION OF THE APPLICATION SYSTEM

5. REQUIREMENTS PREPARATION OF THE APPLICATION SYSTEM

The SDM was driven by a set of functional requirement specifications which possess certain characteristics. As such, the application of the SDM depends on both the existence and appropriateness of the specifications.

First of all, it is clearly necessary that the requirements for the target system be formally stated, i.e., written down, before SDM may be applied. It is not uncommon that requirements for a proposed system are never committed to paper, especially in the case of smaller system or "in house" system development (as opposed to contracted system development). Fortunately, in our case such a document had already been generated. Hereafter we shall refer to this document as the "original system specification" or simply the "original specification" so as to distinguish it from the set of functional requirements prepared for SDM use, which we shall refer to as the "functional requirements set."

Extensive work has been done on user needs analysis for the new test system. The project manager has travelled around the United States and Europe to interview users of the current test system in this effort. The rapidly advancing field of integrated circuit technology has also been studied

in order to predict unforeseen user needs. Based on the results of these studies and their technical experience in the field of PCB testing, designers of the new test system have put many man-years of effort into deciding what features the new system should possess.

The original specifications for the entire system were organized in a "hierarchical form." At the top level, there are two major sections, one for the software subsystem requirements, the other for the hardware subsystem requirements. Requirements for the Test Program Preparation facility occupy one subsection under the software subsystem section. Similarly, these requirements are organized in a "hierarchical" fashion under this subsection.

Based on the original specifications, the author prepared the first draft of the requirement statements appropriate for SDM use. At the outset, guidelines developed during previous SDM applications and experience learned from them were studied to obtain a better grasp of how the set of SDM requirements should be prepared. Huff's clarification of "requirements" and "requirements specifications," and the conceptual framework he put forth for thinking about them within the System Development Cycle [Huff June 1978], were found to be helpful for this process.

5. REQUIREMENTS PREPARATION OF THE APPLICATION SYSTEM

The first draft of requirement statements were then examined by the designer(s) to make additions, corrections, and modifications. This was by no means the final version of the statements. Numerous additional modifications to both statement form and content were made throughout the following stages of the methodology. Certain new statements were deleted or merged together, and minor or major wording alterations were made to some. The final version of the SDM functional requirements set is given in Appendix A. The requirements are constructed in "template forms." The template technique is based on a set of seven basic "requirement statements templates" in which each template corresponds to a general category of statement type (see Figure 5.1). For detailed description of the template approach, see [Huff June 1978]. The template approach was found to be effective; an assessment of it will be found later in the chapter.

5. REQUIREMENTS PREPARATION OF THE APPLICATION SYSTEM

REQUIREMENT STATEMENT TEMPLATES

A. Existence.

There (can/will) be <modifier> <object>

B. Property.

<Mod> <object> <can/will> be <mod> <property>

C. Treatment.

<Mod> <object> (can/will) be <mod> <treatment>

D. Timing.

<Mod> <object> (can/will) <timing relationship> <mod> <object>

E. Volume.

<Mod> <object> (can/will) be <order statement> <index> <count>

F. Relationship.

a. Subsetting.

<Mod> <object> (can/will) contain <mod> <object>

b. Independence.

<Mod> <object> (can/will) be independent of <mod> <object>

Figure 5.1

5. REQUIREMENTS PREPARATION OF THE APPLICATION SYSTEM

Recall that the three important characteristics of a set of SDM requirements are (1) unifunctionality, (2) implementation independence, and (3) common level of generality (abstraction)(*). In the following sections, we elaborate on the meaning of these terms with examples drawn from the current application. We also report the difficulties encountered in ensuring the possession of these characteristics by the functional requirements set. Finally, the template technique will be assessed.

5.1 Unifunctionality -

> Each statement describes a single function (not multiple functions) to be featured in the target system.

The original specification contains some statements that are judged to be multi-functional:

(*) A complete list of the characteristics of "good" requirement statements in the SDM context can be found in Section 3.2.

5. REQUIREMENTS PREPARATION OF THE APPLICATION SYSTEM

> "It is required that fault simulation can be resumed after the simulator is terminated, whether due to fault detection or detection of end of the test program. This will allow the fault simulation to be resumed when more steps are added or to achieve more resolution."

This description specifies two distinct features of the target system:

> (1) Fault simulation can be resumed for more resolution, and
> (2) Fault simulation can be resumed when more test steps are added to the end of the test program.

The two features refer to two different types of resumption. The first type is for the purpose of higher fault resolution. The second type is to eliminate the need of re-simulating the entire sequence of input patterns in case more steps are added at the end of the test program.

In general, the "uni-" part of the "uni-functionality" property is easy to ensure, especially with the template approach. The template approach tends to force statement specifying multiple requirements to be broken into several

brief, one-sentence statements. The "functionality" part of the property is harder to verify. A true functional requirement must be implementation independent; therefore once the "uni-" part of the characteristic is ensured, the problem is reduced to deciding if the requirement statement is implementation independent or not. This issue will be addressed in the next section.

5.2 Implementation Independence -

> Each statement should be implementation free, i.e., ought to specify <u>what</u> is required of the target system but not <u>how</u> that requirement is to be met.

The original specification was found to contain some implementation dependent requirements, for example:

> "The user must be able to override (default) characteristics of tester pins by making suitable entries in %ADAPTOR section in circuit description source file."

The part of the above statement "...by making suitable entries in %ADAPTOR section in circuit description source file" is a procedural statement which indicates how characteristics of tester pins can be overriden. Although to the designer(s), this approach appeared to be the most appropriate, including such procedural information in the functional requirement may constrain later design options at the outset, and result in some potentially superior alternative never being considered. Thus, only the what part of the above statement was included in the functional requirements set:

> Characteristics of tester pins can be overriden by the user.

Although not specified as part of the functional requirement, the procedural information was still included in the specifications as a comment to the above functional requirement. This is simply one of the many cases where the designer(s) requested to have the procedural information (implementation issues) be specified, for they believe these particular implementation approaches are probably the most appropriate ones and would like to see them be reflected in the requirement statements. In cases like this, a comment

section is added to the requirement statement. Within the comment section, the designer(s) may formally include any additional information which is not appropriate or relevant for the basic requirement statement.

The same phenomenon -- the designers were reluctant to leave out certain implementation issues -- was observerd in the SDM application to the new M.I.T. Budgeting System [Huff June(a) 1979] . Such a phenomenon should not be surprising, for in the real world people are very much concerned with how to get things done. Considering implementation issues during the functional development phase is necessary for the purpose of ensuring the feasibility of a functional requirement; however, the danger involved here is that one often becomes so immersed in a particular implementation approach that he confuses functional issues and implementation issues. The SDM approach to requirements preparation constantly reminds the designers of the true functional requirements.

In the case of the current SDM application, some functional requirements are not explicitly stated in the original specifications. Instead, their associated implementation issues are stated. In cases like this, the specific functional requirements have to be elicited by studying what the implementation aims to support.

In some cases, deciding whether a statement is implementation independent or not is difficult, for the decision may depend on the level of generality at which the set of requirements are meant to be specified. This issue will be addressed in the next section.

5.3 Common Level of Generality -

> The set of requirements should be at a common level of generality (abstraction), to the extent possible.

In the original specification, some requirements concerning the Circuit Description Language (CDL) were specified as follows:

> "... Each statement in the %CIRCUIT section shall consist of a name, type, and connection field where one or more spaces, tabs, or carriage returns terminate name and type fields. ...
> The name field is the first field in a statement. It contains the name assigned to an IC, or a special device to be coded into the circuit

description. The name field shall follow the convention of the Circuit Description Language used in the current test system with the following exceptions: (a) names for external paths are optional, (b) .... (c) ....

Statements in %ADAPTOR section map test pin numbers to intermediate node names or externals used in %CIRCUIT section. The allowed test pin types are D for digital (driver/sensor) and A for analog. If no test pin type is specified, D is assumed...."

These statements were judged to be at a too detailed level. They can be re-stated at a higher abstract level (expressed in the template form):

(1) Unique names for each component on the board will be specified.
(2) The type of each component will be specified.
(3) Interconnections between components will be specified.
(4) Symbolic names for external signals can be specified.
(5) Each tester pin type will be specified.
(6) Interconnections between the board and tester

pins will be specified.

.

.

These requirements can be stated at a even higher level:

(1) Description of the board will be specified.
(2) Description of the tester pins will be specified.
(3) Description of the interface between the board and the tester will be specified.

We chose to state the requirements at a level higher than the one above:

There will be a circuit description language which allows users to specify board-related information.

At a very high level, this requirement can be stated as follows:

Information related to the board will be provided.

Note that this statement does not indicate by which means, e.g. using circuit description language, the board-related information will be provided.

Originally, Andreu included "common level of generality" as a property that the SDM requirements set should exhibit for the purpose of proper set decomposition. Andreu claimed that requirements not meeting this property could create problems [Andreu and Madnick, December 1977]:

> ".... For example, numerous detailed requirements can be assessed to be interdependent with another of broader scope, thus bringing them all closer, for set decomposition purposes, than they would otherwise be. This can cause the broader requirement to be assigned to the same subset as narrower ones, possibly neglecting its interdependencies with others of similar (broad) scope."

As far as SDM requirements preparation is concerned, this property is important in that often decisions concerning whether a requirement is implementation independent or not depend on at which level of generality we want to specify the

functional requirements. For example, one can argue that one of the above statements:

> There will be a circuit description language which allows users to specify board-related information.

is merely a particular implementation approach to the other statement specifed at a higher level:

> Information related to the board will be provided.

The rationale behind this argument is that there can be ways other than using a circuit description language to provide board description to the test system, e.g. having the system directly read the schematics of the board. Thus we see how a level of generality must be chosen first before some decisions on "implementation independence" can be made.

A set of requirements may satisfy the "uni-functionality" property, i.e., each statement states only one function, but are specified at a wide range of levels. In this case, again a proper level of generality must be chosen first; then each requirement statement will be examined, and modified if necessary, so that it is stated at the chosen level. If a requirement is specified at a too

high level, one should try to break it down into a few specific statements each of which specifies a single sub-function. If a requirement is specified at a too low level, one should try to elicit the motivation behind this requirement and re-state it at a higher level. Sometimes a group of lower-level requirements statements can be recognized as sub-functions for some specific function at the chosen level, in this case these statements should be replaced by a proper one that states the specific function.

To state all the requirements at a common level of generality is not easy. The first difficulty is concerned with the choice of an appropriate level. Secondly, as no definitions exist for "various levels of generality," determining if a requirement is stated at a particular level could be a difficult task. Lastly, even if a requirement is found to be stated at an inappropriate level, often it is neither proper to exclude it from the requirements set, nor is it possible to re-state it at a level common to the chosen one. Due to these difficulties, we can only try to state the requirements at a common level of generality to the extent possible.

5. REQUIREMENTS PREPARATION OF THE APPLICATION SYSTEM

5.4 ASSESSMENT OF THE TEMPLATE APPROACH

The template approach is not intended as a way to automate, or even mechanize, the creation of SDM requirements (a task which is surely impossible at this stage in the development cycle). Rather, it is intended simply to structure and guide the thought processes involved in carrying out part of the architectural design activity. Experience gained from using the template forms to prepare the requirements for the current SDM application leads to the following assessment of this approach.

The template approach provides a way of better understanding the requirements. By working the requirements statements into template technique, one is forced to consider exactly what each statement means, and how it ought to be expressed in terms of templates. With little practice, statements or parts of statements that are ambiguous or unclear tend to stand out, as they tend to obstruct the transformation of the statement into a template form.

The template technique encourages the generation of "uni-functional" requirements in that a requirement statement which specifies more than one thing must be split into two or more pieces in order to map it into template form. The

template approach also helps to detect procedural statements. Procedural information is likely to be stated in a paragraph rather than one simple sentence, for usually one sentence is not enough to specify "how" things are to be done. Trying to translate a lengthy statement into template form makes one suspect the nature of the statement -- is it a functional statement or a procedural statement? In general, with each requirement expressed in simple template form, it is easier to check if requirements contain the desirable characteristics for the purpose of SDM application.

The template form was found to be flexible and easy to use. More than one form may be suitable for stating a requirement. For example, the following two statements

> Test program developed on the current system can be converted to new format.

and

> There will be Current-to-New test program conversion facility.

request for the same conversion tool.

The designer who formally wrote up the original system specifications has made the following comment on the template approach:

> "The template technique should have been adopted in specifying the system requirements at the very beginning. Regardless of SDM application, the major advantage of the template approach is that it allows one to concentrate on what to put down as system requirements rather than how to nicely put these requirements together in prose."

This author has found that no great additional effort is required to translate the original specification, stated in a more usual form -- an English-prose description -- to template form. If this is found to be true in the general case, the universality and usefulness of the SDM approach will be substantially enhanced.

5. REQUIREMENTS PREPARATION OF THE APPLICATION SYSTEM

5.5 SUMMARY

Immediate design benefits have resulted from the excercise of SDM requirements preparation. One benefit lies in the simple fact of having to think carefully and repeatedly about what each requirement really means, and what the true functional requirements really are. The set of functional requirements statements generated as a result is a valuable document in itself. Its value lies in the following facts. First, the set of functional requirements are explicitly separated from the procedural information (implementation issues). Second, each of these requirements is stated in a brief, one-sentence fashion which is easy to read. Third, the requirements are stated at a common level of generality which is easy for the user to understand.

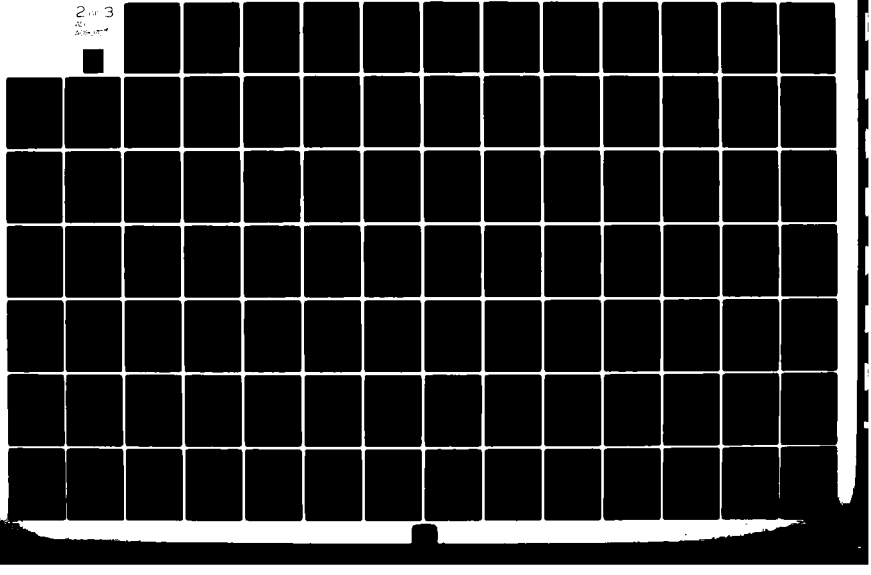
6. INTERDEPENDENCY ASSESSMENT OF THE APPLICATION SYSTEM

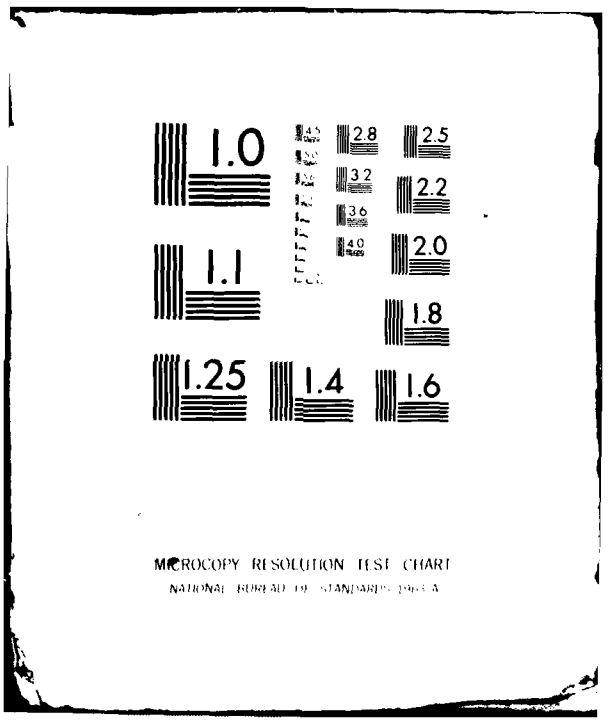
Once the requirement statements had been expressed in a form appropriate for SDM use, work began on determining the existence and weight of the implementation interdependency for every pair of requirements.

Recall that since the goal of the methodology advocated here (SDM) is the identification of design subproblems, the interdependencies defined among requirements should be consistent with that goal. At this point, the reader is recommended to review Chapter 2 and the Section 3.1.2 to ensure that he is familiar with the notion of "interdependency" and the purpose for performing the interdependency assessment activity.

The interdependency assessment for the current SDM application was performed by the chief designer, monitored by the author, and examined by another designer. Throughout the interdependency assessment activity, additional modifications to the set of requirement statements were made. Certain new statements were added in light of improved understanding that occurred as a result of discussion. Some statements were deleted or merged together, to make the set of requirements more appropriate for SDM use. All the interdependencies identified and weights assessed were recorded. The final

AD-A090 707 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
A SYSTEMATIC APPROACH TO COMPLEX SYSTEM DESIGN: AN APPLICATION --ETC(U)
MAY 80 P TUNG N00039-80-K-0573
UNCLASSIFIED CISR-P010-80002-13 NL




1.0
1.1
1.25
1.4
1.6
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

version of interdependency statements is presented in Appendix B. Each statement indicates the two requirements involved in the interdependency, the weight, and a description of the interdependency.

It is important to realize that this step is the most creative one on the part of the designer in the context of SDM. The designer's expertise, past experience, and intuition can play a central role in this activity.

In the rest of this chapter, we will discuss how the existence and weight of an interdependency can be determined, with examples drawn from the current SDM application. Also addressed in this chapter is the issue of time involved in performing interdependency assessment. In the last section, some summarized comments concerning this methodological step are presented.

6. INTERDEPENDENCY ASSESSMENT OF THE APPLICATION SYSTEM

6.1 DETERMINING THE EXISTENCE OF AN INTERDEPENDENCY

For each pair of requirements, the designer considered whether or not significant implementation interdependencies existed between the two requirements. The basic approach to determining the existence of an interdependency is as follows:

(1) One first thinks about how one of the requirements would most likely be implemented. This generally requires thinking through some detailed design or procedural-type issues.

(2) With that "conceptual model" in mind the same thing is done as regards the other requirement.

(3) Then the two conceptual models of implementations are jointly compared to determine whether

(a) One scheme makes it easier to implement the other (a supporting type of interdependency);

(b) one scheme makes it harder to implement the other (a conflicting type of interdependency);

(c) There is no appreciable overlap, either in a supporting sense or in a conflicting sense, between the two.

The result of this comparison suggests the existence or non-existence of an interdependency between the requirements under consideration.

(4) If an interdependency is identified, a one-sentence description is written down to serve as a brief document of the interdependency. The purpose of this description is two-fold. First, it helps one remember why the interdependency was made; second, it briefly explains to others what the interdependency is.

The designer should repeat this process, to the extent possible, by considering more potential implementation schemes.

Some informal guidelines for generating the conceptual models based on which both supporting type and conflicting

type of interdependencies can be made were developed by Andreu [Andreu November 1977]. These guidelines were found helpful in performing the interdependency assessment activity. Two examples, drawn from the current SDM application, are presented below to illustrate how supporting and conflicting interdependencies can be identified.

Given a pair of requirements, a conflicting interdependency can be identified in case one requirement poses additional burden on the other. An example is the pair of requirements 45 and 54:

> "There will be a good board simulator," and
>
> "Properties of a particular target tester will be taken into account by the simulator."

The simulator's ability to take properties of a particular tester -- the one on which the boards will be tested -- into consideration for simulation is an additional burden posed by requirement 54. Many examples of such conflicting interdependencies were identified for the current SDM application.

6. INTERDEPENDENCY ASSESSMENT OF THE APPLICATION SYSTEM

Two requirements may call for some similar type of support; this common support may be provided to both requirements by an particular implementation scheme in the eventual system. An example is the pair of requirements 48 and 59:

> "Results of good board simulation can be properly displayed," and
>
> "Internal faulty activity for an undetected fault can be displayed to user."

Both requirements call for a display facility for the internal nodal values recorded throughout the simulation. These two requirements are interdependent in a supportive sense because implementing one such display facility in the eventual system could satisfy the two requirements simultaneously.

Identifying interdependencies was found to be somewhat harder to perform than expected. Namely, it was difficult for the designer to constantly keep in mind what interdependency assessment was supposed to be. Interdependency assessment is intended to identify interdependencies at the implementation level. However, what

often happened during interdependency assessment was that the focus tended to shift from issues concerning whether or not the two requirements were related at the implementation level to whether they were logically related. This can be seen with the example of requirements 21 and 22:

> "Default driving characteristics of output or bidirectional pins for each specific tester will be supported," and
>
> "Default driving characteristics of output or bidirectional pins for each specific tester can be overriden by user."

At the first glance, it might appear that since both requirements concern "default driving characteristis of output pins or bidirectional pins," they must be interdependent. This is an instance of what is meant by "logical relationship" above. Since this kind of logical relationship is easier to identify than the implementation interdependency, it tends to occur to the designer's mind first and thus obscure the search for true implementation interdependencies. To overcome this problem, throughout the interdependency assessment for the current SDM application

system, the author constantly reminded the designer of the true meaning of "interdependency" and questioned the nature of any doubtful interdependency. If an interdependency was found to be something other than an implementation interdependency, the designer was asked to re-assess the interdependency at the implementation level. In the above example, the implementation for requirement 21 involves keeping the default information for each specific tester somewhere in the system and making it available to the simulator; on the other hand, implementation for requirement 22 involves some mechanism by which the user can override the default information; the two requirements do not interact in terms of implementation. It was found that as long as the meaning of "interdependency" was clearly kept in mind, no great difficulty was encountered to generate conceptual models based on which interdependencies could be assessed.

Recall that in preparing the requirements set for the current SDM application, statements in the original specification that were considered to be too general, too detailed, or implementation dependent, were not included as individual requirement statements in the requirements set. Although not included in the requirements set, the ideas embedded in these statements, which were originally generated by the designers based on their past experience, expertise,

understanding of the system objectives and design constraints, did play a central role in interdependency assessmet activity. Statements that were considered "implementation dependent" often suggest conceptual models based on which interdependencies could be assessed. Requirement statements that were judged to be either "too general" or "too detailed" often provide a better understanding of the requirements in the SDM requirements set. Recall that requirements exist at different levels of generality, and those at a higher level usually "drive" those at a lower level. In order to fully understand the meaning of a requirement, it helps if one knows the motivation behind the requirement(*), and the specific sub-functions, if there is any, the requirement intends to contain(**). Interdependency assessment activity requires one to think about how a requirement can be satisfied, and a good understanding of the requirement is essential for performing this task.

(*) The motivation usually is indicated in one of those statements that were considered "too general."

(**) The specific sub-functions usually are stated in those statements that were considered "too detailed."

6. INTERDEPENDENCY ASSESSMENT OF THE APPLICATION SYSTEM

As far as considering alternative implementation approaches is concerned, it again relies on the designer's expertise, knowledge of the currently available techniques, and his creativity. The range of permissible implementation approaches, however, is limited by design constraints. Design constraints exist in a variety of terms, e.g., time constraint (deadline for the system), constraints imposed by the particular computer on which the system is to be implemented, etc. System objectives themselves may imply design constraints. For example, one of the objectives of the new test system claims that the new test system must be compatible with the current test system from the user's viewpoint. This objective of maintaining compatibility imposes some design constraints and thus limits the range of permissible solutions.

6.2 WEIGHT ASSESSMENT AND THE SCALING PROBLEMS

For each interdependency identified, a "weight" must also be assigned to it. Such a weight could be taken to indicate the "strength of interdependency" between a pair of requirements. With this interpretation, two requirements that are seen to be closely related, in implementation terms, would be judged to be "relatively highly interdependent." Alternatively, the interdependency weight could be defined in

a subjective probability sense. That is, the weight would represent "how certain the designer is about the interaction between the two requirements." This interpretation would be consonant with uncertainty inherent in the interdependency assessment process itself. With this definition, a high weight will be assigned to an interdependency if the designer is very certain that the two requirements would be coupled in implementation; whereas if the designer believes that there is only a fairly small possibility of the requirements being coupled, the assigned weight would be lower.

In the current SDM application, for each interdependency identified (with a few exeptions), the designers were usually fairly certain that the requirements would be coupled in implementation. Thus, the weights were almost always assesed based on "strength of the interdependency" only. This can be determined by asking how much harder or how much easier one requirement makes implemention of the other.

The two examples presented in the proceeding section will be used here to illustrate the idea of weight assessment. In the case of requirements 48 and 59:

"Results of good board simulation can be properly displayed," and

"Internal node faulty activity for an undetected fault can be displayed to user;"

the interdependency was judged to be a strong one because the implementations of the two requirements overlap very much. Once one requirement is satisfied, the other can be satisfied with little additional implementation effort. In the case of requirements 45 and 54:

"There will be a (good board) simulator," and

"Properties of a particular target tester will be taken into account by the smulator;"

the interdependency was judged to be a weak one because the additional burden requirement 54 poses on the simulator (requirement 45) is not very much, compared with the entire effort required for the design and implementation of the simulator.

Associated with the interdependency weights are some scaling problems. First, a range must be specified over which weights will be allowed to vary. Although not absolutely necessary, compatibility with the basic model

employed in Phase-I SDM, among other reasons, suggested that link weights be chosen from numerical range [0,1]. Another scaling problem concerns requiring the designer to choose from "pre-set" weight values (e.g., low, medium, high) versus a continuous range of values. If pre-set weights are used, some mapping to numerical values must also be chosen (e.g., "strong" corresponds to numerical weight of 0.8, etc.). All link weights must eventually be encoded numerically, for use in the graph partitioning algorithms.

SDM researchers' experience has indicated that system designers most likely would not require the capability of specifying weights directly in numerical terms [Huff and Madnick July 1978]. The following scheme, using a simple three-way breakdown, was proposed:

Design's Judgement About Weight	Code	Numerical Value
Strong	S	0.8
Average	A	0.5
Weak	W	0.2

This particular encoding of weights achieves several useful results. Firstly, it may be easily interpolated, as shown below:

Weight:	S+	S	S-	A+	A	A-	W+	W	W-
Numerical Value:	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1

Secondly, the fact that all weights fall in the [0,1] range is useful for normalization and later decomposition.

This three-way breakdown approach was used for assessing interdependency weights in the current SDM application. Most of time this approach was found satisfactory.

6.3 EXECUTION TIME FOR INTERDEPENDENCY ASSESSMENT ACTIVITY

Andreu expressed concern over the time required to execute the SDM interdependency analysis on requirement sets of non-trivial size [Andreu 1978]. The interdependency assessment activity for the current SDM application system consumed approximately sixteen hours of meeting time, which is not a inconsiderable load. It was found, through the current SDM application, that the execution time for interdependency assessment largely depends on how "accurate" the designer wants the results of the interdependency

assessment to be. The more discipline the designer has in carrying out the process, the more accurate the results may be. The discipline here refers to the designer's repeatedly asking himself the following questions and working until the answers are satisfactory to him:

(1) For each pair of requirements, have enough implementation alternatives been considered?

(2) If an interdependency is identified, is it really an implementation interdependency? Or is it merely a logical relation?

(3) Are conceptual models used for assessing the current pair of requirements relevant to pairs of requirements for which assessment has already been made?

The more times these questions are asked and answered, the more time is spent executing the interdependency assessment. This activity should not be considered as an additional effort on the part of the designer. In reality, any system designer makes use of such conceptual models, implicitly or explicitly, at some point in the design process; otherwise

one could not end up with a design at all. The methodology advocates working through this process explicitly, in a systematic manner (dealing with a pair of requirements at a time), so as to help the designer to cope with the complexity of the design problem.

6.4 SUMMARY

This exercise indicates clearly that there are immediate design benefits to be had from the SDM interdependency assessment activity. The common source of the immediate design benefits lies partially in the simple fact that this activity requires the designer to carefully and repeatedly consider what each requirement means, how each might be implemented, and how alternative implementation schemes interact, in a structured way.

The chief designer who performed the interdependency assessment made the following comment about this activity:

> "The exercise allows me to recognize which requirements are the features the system aims to provide, which requirements are mainly for the purpose of users'

> convenience or for ease of implementation. Such recognition encourages me to consider other implementation alternatives. .... To faithfully perform the interdependency assessment process is not a easy task; but the entire exercise is worthwhile."

Another designer has made the following remark concerning the explicit links established among requirements as a result of the interdependency assessment activity:

> "An ad-hoc architecture may not be bad at all. What usually causes the problem is the overlooking of the hidden interaction that exists among subproblems in the system architecture. The SDM interdependency assessment activity helps one to identify such interaction; moreover, the resulting interdependency assessment statements serve to reflect all the interaction explicitly."

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

The interdependencies between pairs of requirements that were identified and weighed during the interdependency assessment process define the requirements graph for the current SDM application system. They were entered as input to the decomposition analysis package(*). The facilities of the decomposition package were then used to develop decompositions of the requirements graph, to evaluate them using the objective function M (for the definition of M, see Chapter 3.1.3), and to modify and manipulate the decompositions in various ways.

Each of the five decomposition techniques (four clustering techniques, and the interchange algorithm) were applied to the current SDM application system. These decomposition techniques are briefly described in Chapter 3.2.3; for detailed descriptions, see [Huff January 1979] and [Huff February 1979]. The goal was to produce a decomposition that was the best in terms of identifying high-strength, low-coupling subgraphs (as measured by M). It turned out that a decomposition which was initially generated

---

(*) This decomposition analysis package was developed by Huff for decomposing and analyzing the SDM gragh. The package operates within the VM/370-CMS environment and presently runs on the MIT 370/168 computer system [Huff February 1979].

by HIER3 (one of the bottom-up clustering techniques) and then modified by the author, was judged to be the best (with M = 0.52). The other three bottom-up clustering techniques (HIER1, HIER2, HIER4) produced decompositions with relatively low values of objective function. The interchange algorithm, taking a top-down approach, generated a decomposition which was close to the best (with M = 0.51). In fact, the author used the decomposition generated by the interchange algorithm as a reference to modify the one initially generated by HIER3 to obtain the final best decomposition.

The best decomposition was then studied -- both the requirements subsets and the sets of interdependencies between requirements subsets -- so as to formulate a design framework of the Test Program Preparation System. It was expected that during the study some counter-intuitive results would be identified. It was also expected that these counter-intuitive results, on closer inspection, might either indicate earlier errors in assessments, requirements formulation, etc., or prove to be correct but simply unforseen. It turned out that the latter case never occurred; on the other hand, some errors in requirements formulation and assessments were detected and thus some minor modifications were made accordingly.

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

The modified version of the requirements graph was re-entered to the decomposition analysis package and a second iteration of the decomposition process was performed. The best decomposition obtained this time (again it was generated by HIER3 and then modified by the author) has an objective function of M = 0.69. The graph decomposition was studied to formulate a design framework for the current SDM application system. In the sections that follow, the results of the study are to be discussed.

7.1 ANALYSIS OF THE DESIGN SUBPROBLEMS

A total of eight design subproblems are identified in the best decomposition of the requirements graph. Table 7.1 contains the summary description for each subproblem. Also included in the table are the size of each subproblem and the specific requirements it contains. A listing of the abbreviated system requirements (no Comment sections included there for brevity) organized according to subgraphs can be found in Appendix C. In this section, the central focus of each subproblem and the reason for requirements falling into each specific subproblem will be discussed.

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

Subproblem and No. of requirements	Summary Description	Requirements
1 (8)	Test Program Processing	1,2,7,10, 11,12,13,14;
2 (24)	Circuit Description Coding and Processing	5,6,8,9,15-26, 31,34,36,43,51, 52,68,69;
3 (4)	IC Models Conversion	28,29,41,42;
4 (5)	IC Library Handling	27,30,32,33,35;
5 (13)	Output Patterns Simulation	37-40,44,45,47, 48,50,53-55,59;
6 (4)	Fault Modeling and Insertion	64-67;
7 (9)	Test Program Evaluation and Fault Isolation Preparation	46,49,56,57,58, 60-63;
8 (2)	Test Program Macro Processing	2,3;

TABLE 7.1
Subproblem Summary Descriptions
of the Best Located Decomposition

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

Subproblem 1 -- Test Program Processing

This subproblem contains 8 requirements, including 1, 2, 7, 10, 11, 12, 13, and 14. The central focus of this subproblem is Test Program Processing. At the heart of this subproblem are requirements 1 and 2. Requirement 1 calls for a proper test language definition. The test language should allow the test engineer to control the input test signals and device power for the purpose of testing. Requirement 2 pertains to the test program processor. The processor is responsible for generating the data base needed for simulation and the data base needed for diagnosis. Other requirements in his subproblem are related to these two requirements in various ways.

Requirements 12 and 13 are concerned with allowing multiple sets of test patterns be contained in the same test program. These two are interdependent with each other for they require common implementation support. Requirement 7 requests the capability to incrementally compile the test steps added to the end of a processed test program without re-compiling the entire test program. These requirements more or less impose additional burden on requirements 1 and 2.

This subproblem also includes requirements concerning program conversion facilities. Requirement 11 calls for a facility that converts test programs developed on the current test system to the new format. Requirement 14 requests a conversion facility that converts a GO/NO-GO program generated in the new format to the format compatible with the current test system. These two requirements are to support those current users who will use the new system for test program preparation, but will keep the current testers for testing and diagnosis. They fall into this subproblem firstly because the conversion facility must be aware of the new test program format; secondly, the design and implementation of the test language and the test program format should also take these two requirements into consideration to insure the feasibility of the conversions.

Requirement 10 is concerned with the generation of a GO/NO-GO test program. A GO/NO-GO test program is a program that contains both input patterns (the original user-created program) and output patterns at the proper test step. The reason that this requirement falls into this subproblem is because the test program processor may serve to support part of its implementation.

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

Subproblem 2 -- Circuit Description Coding and Processing

This subproblem is the largest in terms of the number of requirements: twenty four requirements, including 5, 6, 8, 9, 15-26, 31, 34, 36, 43, 51, 52, 68, and 69. The central focus of this subproblem is Circuit Description Coding and Processing.

At the heart of this subproblem are requirements 15 and 19 which correspond to circuit description coding and processing respectively. Requirement 15 calls for a proper Circuit Description Language (CDL) which will allow the user to specify all the board-related information, e.g. the components on the board and their interconnections, tester-board interface description, etc. Requirement 19 pertains to the processing of the circuit description information. It is responsible for generating the data bases required by other tasks involved in test program preparation, and data bases needed by the Guided Probe Facility for the purpose of diagnosis. Most other requirements in this subproblem directly relate to requirements 15 and/or 19.

Requirements 16, 17, and 18 are concerned with various types of partial sharing of circuit description information. Requirement 20 calls for a partial re-processing capability.

This enables the user to modify some information without having to re-process the entire circuit description file. These requirements fall into this subproblem because they should be taken into consideration for the design and implementation of requirements 15 and 19.

Some requirements are placed in this subproblem because they impose additional burdens on requirements 15 and 19. Requirements 5, 6, 21, 22, 31, 36, 51, 52, 68, and 69 are concerned with output pin characteristics for each specific tester type and output pin characteristics for each specific IC family type. The new simulator will take output pin characteristics into account for the purpose of accurately modeling the board activity. These requirements end up in this subproblem because most likely the circuit description processor (requirement 19) will be responsible for providing the simulator with the proper information with respect to output pin characteristics. Furthermore, CDL may have to provide constructs which allow the user to override the default characteristics.

Requirements 8 and 9 are concerned with proper handling of devices that cannot be modeled (e.g. analog devices), and direct manipulation of the internal node values by the user. These two requirements are strongly interdependent with each

other for they require very common support and thus can be satisfied simultaneously. They are interdependent with requirements 15 and 19 in that they impose additional burdens on these two requirements: CDL must provide the user with the proper mechanism, and the processor must provide the simulator with the proper information, in order to realize these two requirements.

Some requirements are interdependent with requirements 15 and 19 in a supportive sense. Requirements 23, 24, and 25 pertain to the reporting of various board-related information that is to the user's interests. These requirements end up in this subproblem because these reports can be generated with no great difficulty after completing a good run through the circuit description processor. Requirement 34 requests a "circuit description" IC modeling technique, an IC modeling technique which models an IC in terms of the basic elements the IC consists of. This requirement ends up in this subproblem because describing a model of an IC in terms of its basic elements is very similar to describing a board in terms of the components on the board (which themselves are IC's and other special devices). The user can treat the basic elements within the IC as if they were IC's on the board. Effort must be put into the design of CDL and CDL processor so that they can be used for both board modeling

and IC modeling. Requirement 43 pertains to Conditional Connectivity Language (CCL) which allows the user to define connectivity of an IC model. Connectivity information, which indicates for each specific output of the IC, the subset of the inputs that can affect this output, is needed to support Guided Probe Facility. The reason why this requirement ends up in this subproblem is that a syntax similar to that of the CDL can be used for CCL.

Requirement 26 pertains to the conversion of the circuit description source files on the current test systems to the new format. This is to support users of existing test systems who are to use the new test program preparation facility. This requirement falls into this subproblem for two reasons. Firstly, the conversion facility concerns the new format of the circuit description file. Secondly, this conversion requirement should be taken into consideration for the design and implementation of the circuit descripton file format to ensure the feasibility of the conversion.

Subproblem 3 -- IC Models Conversion

This subproblem includes four requirements, including 28, 29, 41, and 42. All of the four requirements directly relate to IC Models Conversion. A tremendous amount of effort has been put into the development of the existing IC libraries, both the system library and many user libraries. The objective here is to fully utilize previous work so as to reduce the effort to re-build the libraries. The major effort involved in building an IC library is modeling the IC's. Conversion tools will be provided so that individual IC models built on the current system can be converted to models represented in the new format. Requirements 41 and 42 call for IC models conversion tools for two different types of IC models. Implementation for these two requirements will be reduced to a source-to-source conversion effort with the support of requirement 29, which provides the basic elements that have been widely used for building currently existing IC models. The inclusion of 90% of the existing system library (requirement 28) is required because IC models supported in the system library are widely utilized by current users to model their own IC's. This requirement ends up in this subproblem because its implementation will be greatly simplified if the other three requirements in this subproblem are properly satisfied.

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

Subproblem 4 -- IC Library Handling

This subproblem contains five requirements, including 27, 30, 32, 33, and 35. The central focus of this subproblem is IC Library Handling. Requirements 27, 32, and 33 ending up in the same subproblem makes good sense since their implementations are all concerned with the organization of and access to libraries. Requirements 30 and 35 pertain to modeling of ROM's and PLA devices respectively. These two requirements end up in this subproblem because ROM's and PLA devices are two types of IC models to be contained in the library and thus are interdependent with the other three requirements in this subproblem.

Subproblem 5 -- Output Patterns Simulation

This subproblem contains thirteen requirements, including 37--40, 44, 45, 47, 48, 50, 53, 54, 55, and 59. The central focus of this subproblem is Output Patterns Simulation.

At the heart of this subproblem is the simulator itself (requirement 45). A few new "sub-functions" are embedded in this requirement, e.g. better timing analysis will be provided by the new simulator, high impedance state (Z state)

will be simulated (for recent popular use of bus type devices), etc. Requirement 47 is concerned with the speed of the simulator; its being in this subproblem is only natural because the realization of this requirement strongly depends on the design and implementation of the simulator. Other requirements in this subproblem directly relate to requirement 45 in various ways.

Requirements 50, 53, 54, and 55 represent features that ensure an accurate modeling of the board activity. They are interdependent with requirement 45 in that they impose additional burdens on the simulator.

Requirements 37, 38, 39, and 40 are concerned with high-level functional IC modeling and the language used to do the modeling (SCL -- Simulation Command Language). High-level functional IC modeling offers the only viable solution to the problem of modeling complex IC device [Grason 1977] [Armstrong and Woodruff 1977]. The word "functional" refers to the internal function within the chip (from any input change to any resultant output change). To functionally model an IC, procedures which describe the function of the IC must be written by the IC modeler. SCL is a high-level language used to write the procedures. Some of these requirements reflect the inadequacies of the SCL used

on the current system. These requirements end up in this subproblem because the procedures will run as subroutines of the simulator, and additional effort must be put into the design of the simulator to properly satisfy these requirements.

Requirement 44 pertains to the automatic simulation of the internal circuit activity of an IC that has been modeled using circuit description technique (see Subproblem 2 for circuit description IC modeling technique). The purpose here is to automatically generate the connectivity information of the IC (see requirement 43 in Subproblem 2 for the use of connectivity information). This requirement ends up in this subproblem because the same simulator used for board simulation (requirement 45) can be used to carry out connectivity simulation.

Finally, Requirement 48 requires a nodal-status listing facility, i.e., the display of the internal nodal values recorded throughout the simulation. This requirement falls into this subproblem mainly because its implementation depends on what kind of information is generated throughout the simulation, and how the information is recorded by the simulator. The presence of requirement 59 -- display of internal nodal faulty activity for an undetected fault -- in

this subproblem was not initially obvious to the author. At first, the author wondered why it did not fall into subproblem 7 -- test program evaluation and fault isolation preparation. (This is another instance of how logical relationship tends to occur to one's mind first.) The question was raised to the designer who performed the interdependency assessment activity. From his viewpoint, the presence of requirement 59 in this subproblem rather than in subproblem 5 makes good sense; the rationale for it is that once requirement 48 (display facility for internal nodal values recorded throughout the simulation) is implemented, requirement 59 can be satisfied with little additional effort. It makes no difference to the display facility whether the internal nodal values to be displayed are resultant from simulation of a good circuit or simulation of a faulty circuit.

Subproblem 6 -- Fault Modeling and Insertion

This problem contains four requirements (64-67). Fault models must be built before they are inserted into a good circuit for fault simulation. The central focus of this subproblem is Fault Modeling and Insertion. Requirements 64, 65, and 67 are interdependent with each other in that implementations of these requirements all involve fault

modeling mechanism. Requirement 66 is in this subproblem because it represents a constraint which largely eases the implementation of these requirements.

Subproblem 7 -- Test Program Evaluation and Fault Isolation Preparation

This subproblem has nine requirements, including 46, 49, 56, 57, 58, 60, 61, 62, and 63. Its central focus corresponds to Test Program Evaluation and Fault Isolation Preparation. At the heart of this subproblem is requirement 46 -- fault simulation activity. In a simulator-based test program preparation system, fault simulation activity supports both test program evaluation and fault isolation preparation (as described in Section 4.2.4 and Section 4.2.5). Other requirements in this subproblem directly relate to this requirement in various ways.

Requirements 57, and 58 are concerned with reporting of test program quality evaluation. Faults undetected (requirement 57) and percentage of faults detected (requirement 58) must be reported to the user. These two requirements are interdependent with requirement 46 in a supportive sense: they can be easily satisfied as a result of fault simulation activity.

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

Requirements 49, 60, 61, 62, and 63 are all interdependent with requirement 46 in that they represent fault simulation related features which impose additional burdens on the design and implmentation of fault simulation.

Requirement 56 concerns the speed of fault simulation activity. Fault simulation has been the bottleneck for the test program preparation process; this is due to the large number of faults which must be simulated. In addition to the necessity of a faster simulator (requirement 46 in Subproblem 5), other techniques must be employed to reduce the time consumed in the fault simulation activity.

Subproblem 8 -- Test Program Macro Processing Facility

This subproblem only contains two requirements, and is rather easy to interpret. The two requirements (requirements 3 and 4) are concerned with test program macro processing capability. It was argued during the requirements preparation process whether these two requirements should be included or not. The reason for not including them was that a general purpose macro pre-processor would be sufficient for satisfying these two requirements. They do not seem to be interdependent with any other requirement in terms of

implementation and thus they should not be included in the SDM requirements set (the SDM requirements set contains those requirements that are revelant to interdependency assessment only). Nevertheless, the designer who performed the interdependency assessment decided to include them for he did envision a weak interdependency between requirement 3 and requirement 1 (the test language): in order to successfully implement a test program macro processing facility, one should take test language into consideration. These two requirements (3 and 4) forming one subproblem makes good sense in that their implementation is largely independent from other requirements. Their weak interaction with the rest of the system requirements is reflected in the weak link between requirement 3 and 1 (contained in subproblem 1).

* * *

This concludes the analysis and interpretation of the eight identified system design subproblems. In the next section we will briefly analyze the interrelationships (sets of interdependencies) between the subproblems.

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

7.2 ANALYSIS OF SUBPROBLEMS INTERRELATIONSHIPS

There are a total of 13 links interconnecting the 8 design subproblems in the design framework of the current SDM application system. A listing of the set of interdependencies between each identified pair of subproblems is presented in Table 7.2 Some statistics regarding these links are shown in Table 7.3. Since both the number of interdependencies as well as interdependency weights are important determinants of link strength, Table 7.3 also shows total weight for each link. This is just the sum of the weights on all the interdependencies making up each link. In this section, we will provide a brief interpretation of the nature of each subproblem linkage. Summary descriptions of the 13 inter-subproblem linkages are given in Table 7.4 at the end of this section.

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

SUBPROBLEMS PAIR	INTERDEPENDENT NODES
1 -- 2	1-8 ,W 1-9 ,W 1-15,W 2-8 ,W 2-9 ,W 2-19,W
1 -- 3	** NONE **
1 -- 4	** NONE **
1 -- 5	1-45,W 1-53,W 1-55,W 2-45,W 2-48,W 2-53,W 2-55,W 10-45,W 10-55,W
1 -- 6	** NONE **
1 -- 7	2-46,W 7-49,W 10-49,W
1 -- 8	1-3, W
2 -- 3	26-41,S
2 -- 4	15-30,W 15-35,W 19-27,W 19-32,W 19-33,W 20-27,W 27-34,A 32-34,A 33-34,A
2 -- 5	8-45,W 9-45,A 15-55,W 19-45,A 19-53,A 20-45,W 34-44,A 34-45,W 34-53,W 34-55,W
2 -- 6	34-67,A
2 -- 7	19-46,A 20-46,W
2 -- 8	** NONE **

3 -- 4	** NONE **		
3 -- 5	37-42,A		
3 -- 6	** NONE **		
3 -- 7	** NONE **		
3 -- 8	** NONE **		
4 -- 5	** NONE **		
4 -- 6	** NONE **		
4 -- 7	** NONE **		
4 -- 8	** NONE **		
5 -- 6	37-67,A	38-65,A	38-67,W
	40-67,W	53-64,W	54-64,W
	54-65,W	55-64,W	55-65,W
5 -- 7	38-46,W	40-46,W	40-49,W
	45-46,A	45-49,W	48-49,W
	55-46,W	55-49,W	55-60,W
	55-62,W	59-46,A	59-61,W
5 -- 8	** NONE **		
6 -- 7	46-64,S	46-65,S	46-66,A
	46-67,A		
6 -- 8	** NONE **		
7 -- 8	** NONE **		

Table 7.2
Interdependencies Between Requirements Subsets
In Best Decomposition

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

ID No.	Subproblems Pair	Number of Linking Interdep.	Total Weight	Average Weight
1	1 -- 2	6	1.2	0.20
2	1 -- 5	9	1.8	0.20
3	1 -- 7	3	0.6	0.20
4	1 -- 8	1	0.2	0.20
5	2 -- 3	1	0.8	0.80
6	2 -- 4	9	2.7	0.30
7	2 -- 5	10	3.2	0.32
8	2 -- 6	1	0.5	0.50
9	2 -- 7	2	0.7	0.35
10	3 -- 5	1	0.5	0.50
11	5 -- 6	9	2.4	0.27
12	5 -- 7	12	3.0	0.25
13	6 -- 7	4	2.6	0.65

Table 7.3 Statistics for Inter-Subproblem Linkages

Link 1: (Subproblems 1 and 2)

This link contains six weak interdependencies. Two interdependencies pertain to circuit-related information needed for test program processing (1-15, 2-19). The other four are concerned with proper test langauge constructs that allow the user to directly manipulate the internal nodal values (1-8, 1-9, 2-8, 2-9).

Link 2: (Subproblems 1 and 5)

This link contains nine weak interdependencies. Eight of them pertain to information specified in the test program that is needed for simulation: the sequence of test steps which the simulator will follow, and other information concerning delay, bus direction, synchronization, etc., which the simulator will take into account. One interdependency pertains to the simulated output patterns that are to be incorporated into the GO/NO-GO test program preparation (interdependency 10-45).

Link 3: (Subproblems 1 and 7)

This link contains three weak interdependencies. One interdependency pertains to test program information needed for fault simulation activity. The other two are concerned with proper cooperation among incremental test program compilation, incremental simulation, and incremental GO/NO-GO test program generation, for the purpose of preparing the test program in increments through interaction with the simulator.

Link 4: (Subproblems 1 and 8)

This link contains only one weak interdependency. It refers to taking test language into consideration for the implementation of the test program macro processor facility.

Link 5: (Subproblems 2 and 3)

This link contains one strong interdependency. It pertains to the common current-to-new conversion facility for both board circuit description source files and IC circuit description source files.

Link 6: (Subproblems 2 and 4)

This link contains nine interdependencies (three average ones and six weak ones). Three interdependencies pertain to sharing of common IC library access tools (19-27, 19-32, 19-33). One interdependency is concerned with proper modification of IC models in an IC library as a result of partial re-processing (20-27). Two interdependencies pertain to using CDL to specify contents of ROM models and PLA devices (15-30, 15-35). Finally, three interdependencies (27-34, 32-34, 33-34) are concerned with containing Circuit Description type IC models in IC libraries.

Link 7: (Subproblems 2 and 5)

This link contains ten interdependencies (six weak ones and four average ones). One interdependency pertains to automatically simulating connectivity information for Circuit Description type IC models (34-44). Two interdependencies ensure that the mechanism used for direct manipulation of internal nodes is consistent with charateristics of the simulation. The rest of the interdependencies pertain to circuit-related information required for the circuit simulator.

Link 8: (Subproblems 2 and 6)

This link contains one average interdependency. It pertains to using Circuit Description IC modeling techniques to model internal LSI faults.

Link 9: (Subproblems 2 and 7)

This link contains two average interdependencies. The focus of this linkage concerns circuit-related information needed for fault simulation process.

Link 10: (Subproblems 3 and 5)

This link contains only one average interdependency. It pertains to conversion of functional IC models. The conversion facility must be aware of the new SCL definition; furthermore, the design and implementation of the new SCL must also insure that the conversion is feasible.

Link 11: (Subproblems 5 and 6)

This link contains nine interdependencies (two average ones and seven weak ones). Of the nine interdependencies, four pertain to using functional IC modeling technique to support user inserted faults and internal LSI faults

modeling. The other five interdependencies focus on accurate modeling of faulty circuit (good board with fault models inserted in it).

Link 12: (Subproblems 5 and 7)

This link has twelve interdependencies (ten weak ones and two average ones). The relatively large number of interdependencies in this link is due to the fact that the simulator is used for both good circuit simulation and fault simulation (specifically, interdependency 45-46). The central focus of this link is on common simulation issues. Some interdependencies concern the complication brought up by incremental simulation (49-40,45,48,55); some concern the complication brought up by better synchronization between the tester and the simulator (55-46,49,60,62). Three interdependencies pertain to support provided to the SCL (Simulation Command Language) by fault simulation (38-46, 40-46, 40-49). One interdependency is concerned with common fault selection mechanism (59-61). One interdependency is concerned with information, recorded during fault simulation, that is relevant to the display of internal node faulty circuit activity.

Link 13: (Subproblems 6 and 7)

This link contains 4 interdependencies (2 strong ones and 2 average ones). The central focus of the link is on the use of fault models in fault simulation. The link ensures that insertion of fault models is consistent with simulation machinary (46-66), fault simulation provides support to modeling of internal LSI faults (46-67), and finally, that fault analysis for each fault model is properly done prior to fault simulation activity.

* * *

This completes the description of the individual linkages between the design subproblems. Summary descriptions of the 13 inter-subproblems linkages are given in Table 7.4.

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

ID	Subgraphs	Summary Description
1	1,2	circuit description information relevant to test program processing; test language constructs for manipulation of internal nodal values.
2	1,5	test program information relevant to simulation; simulated output patterns for GO/NO-GO test program generation.
3	1,7	test program information relevant to fault simulation; incremental program preparation issues.
4	1,8	concern of test language syntax for test program macro processing facility.
5	2,3	common conversion facility for existing circuit description files.
6	2,4	common access tools to IC libraries.
7	2,5	circuit description information required for simulation; automatic connectivity simulation.
8	2,6	use of the circuit description IC modeling technique for fault modeling.
9	2,7	circuit description information required for fault simulation process.
10	3,5	SCL models conversion.

11	5,6	SCL's support to fault modeling; accurate modeling of faulty circuit.
12	5,7	common simulation issues.
13	6,7	use of fault models for fault simulation; preparation work concerning fault models before fault simulation process.

Table 7.4

Summary Descriptions of the 13 Inter-subproblem Linkages

7. A DESIGN PROBLEM FOR THE CURRENT SDM APPLICATION SYSTEM

7.3 SUMMARY

While the decomposition process supports the architectural design phase by clustering the global system requirements into subproblems, it does not purport to provide the best answer since the techniques are satisfying rather than optimizing. In the case of the current application study, the subproblems and linkages presented in the design framework of the new test program preparation system are quite clear and easy to interpret. All the subproblems were found to have an obvious design focus and the important design implications of the various inter-subproblems linkages were easily identified. Judged by this mixture of intuitive and explicit measures, SDM functioned well in guiding us to the identification of a good design framework.

Overall, among the five decomposition techniques (four bottom-up clustering techniques and one top-down partitioning technique), the top-down interchange partitioning technique appears to be the most promising in terms of producing satisfying results with the least amount of effort from the user. More about the decomposition techniques and the analysis packages will be addressed in the next chapter.

8. Areas for Further Research and Improvement

The original objective of this thesis is to test the applicability of the SDM to software system architectural design. The SDM application to the test program preparation system was quite successful, as far as it went. SDM "worked" in a technical sense, and it was perceived to be useful by (presumably unbiased) system designers who participated in the SDM application. The purpose of this chapter is to take a retrospective view of the current application and discuss the areas which deserve some attention for further study. While there are undoubtably many avenues for further research in the SDM and related areas, to limit the scope of this brief discussion, we will address only those areas which we believe to be most important based on the experience of the current application study.

8.1 The Common Level of Generality Issue

As was mentioned in Section 5.3, it was difficult to frame all the requirement statements at a common level of generality. Indeed, starting from the very first methodological step, the greatest expenditure of time and energy for the current SDM application seemed to be on

dealing with the "common level of generality" issue. Apparently, this is an area that deserves further study. The following questions ought to be considered:

(1) Which level is appropriate for specifying the SDM requirements?

(2) Given that a level is chosen, to what extent can all the requirements be specified at that common level?

(3) From the viewpoint of applying the SDM, is the above extent good enough to avoid undesirable impact on the decomposition process?

During the SDM requirements preparation process for the current application, the designers and the author occasionally found it necessary that requirements be stated at various levels of generality: the general requirements serve to indicate the origin of some detailed ones, and the detailed ones serve to elaborate the general ones. Thus, based on the experience learned from the current application, it seems to be impractical to state all the requirements at

only one common level of generality. Thus, it is advisable to at least allow requirements to be specified at various levels of generality. Then, for the purpose of applying SDM, one may choose a particular level and regard requirements specified at that level as the "requirement nodes" of the SDM graph model. In fact, as was already mentioned in Section 6.1, the presence of those either "too general" or "too detailed" requirements is helpful for performing the interdependency assessment because they provide a better understanding of the requirements in general.

Recall that during the Phase-I methodology development stage, Andreu identified a list of characteristics which the set of SDM requirements should exhibit (see Section 3.2.1). The need for the inclusion of each of these characteristics was justified based on the goal of the methodology [Andreu November 1977]. It is important to note that the "common level of generality" property was not included in this initial list. Rather, it was added to the list later based on a lesson learned from the SDM application to the design of a DBMS -- the first SDM testing case [Andreu December 1977]. The lesson indicated that for the purpose of proper set decomposition, all the SDM requirements should be specified at a common level of generality (see Section 5.3 in this thesis for the quote from Andrue's report). In other words,

unlike those included in the original list, this particular property is basically concerned with proper set decomposition.

This fact raises the question: how about removing the "common level of generality" restriction, i.e., how about allowing requirements to be specified at various levels of generality (*). Perhaps some kind of "hierarchical structuring" mechanism should be incorporated into SDM to support this - another area for further research which is to be discussed more below.

8.2 Study of Hierarchical Structuring Principle

While the motivation behind incorporating hierarchical structuring into SDM is to allow requirements to be specified at various levels of generality, more research work is required concerning the following:

(1) how should hierarchical structuring be incorporated into SDM, and

---

(*) we may still want to restrict the range of levels for design purpose

(2) whether it indeed aids designers in coping with the complexity of the system design problem.

It is well known that hierarchical structuring is a very common and powerful way for humans to deal with complex system (e.g. see [Simon 1969]). In the case of the current SDM application, the original requirements specifications were organized in a hierarchical fashion: the top level sections contain the general requirement statements, and under each of these sections, more detailed requirements were specified. As was observed in the other real world SDM application, the hierarchical structuring approach was also used by the system designers to specify the requirement statements [Huff June(a) 1979]:

> ".... Following initial discussions with the Budget System designers, it was decided that the designers would prepare an initial requirements list,..... This initial set of requirement statements proved somewhat inappropriate for SDM use for various reasons. The most important difficulty concerned the manner in which many of the statements had been constructed by the designers....., many statements consisted of a very general leader statement,

followed by a series of sub-statements..."

In order to support the incorporation of hierarchical structuring, the succeeding methodological steps (i.e., interdependency assessment, decomposition process, etc.) should be extended accordingly. Recall that during the Phase-II SDM development stage, Huff proposed a variety of extensions to the basic binary model (see Section 3.2.3); among them are "hierarchical implication relationships." While being applied to a subset of DBMS requirements for experimental purposes, the hierarchical relationships turn out not to buy the designers much [Huff July 1978]. This is because the requirements were already framed at a common level of generality. If the "common level of generality" is not a restriction for requirements specifications anymore, it is certainly worth re-examining the hierarchical implication relationships.

Finally, proper decomposition techniques must be developed to support incorporation of hierarchical structuring into SDM.

8.3 Techniques for Using the Methodology

Another aspect of further study concerns techniques for using the methodology; in particular, techniques for carrying out the interdependency assessment activity.

In the current application study, two analysis methods were tried: (1) individual, and (2) individual followed by group; although no attempt was made to monitor the relative effectiveness of the different approaches. The author's opinion on this is that the second approach may be better in terms of obtaining a convergence of design opinion less susceptable to bias (i.e., regarding the existence or strength of interdependencies).

8.4 Analytical Techniques for the Decomposition Process

Based on the experience learned from the current application study, the important analytical techniques for applying the SDM concept are reasonably well developed at this point. Nevertheless, from the viewpoint that the decomposition process is the methodological step intended to be automated the most, i.e., the "best" decomposition should be generated with as little amount of effort contributed from the designer as possible, more work is required to automate

this process to the fullest extent possible. As was pointed out by Huff, there are certain graph decomposition techniques that have not been tried in the SDM context; among them, McCormick's approach appears especially promising (see Chapter 5 in Huff's Doctoral Thesis [Huff June(b) 1979]). It is likely that additional study of the clustering and network literature would unearth more potentially useful approaches to the decomposition problem.

Another area that deserves attention concerning the decomposition process is the "friendliness" of the environment in which the designer and the decomposition techniques cooperate to generate the best result. Of course, if the decomposition techniques are intelligent enough to efficiently produce the best result with no human interaction at all, little has to be done concerning the "human interface" issue. As long as there involves human interaction, it is strongly recommended that a friendly interface be provided. An unfriendly interface not only would hinder the popularity of the SDM, but more importantly, it simply defeats the purpose of the methodology. For after all, the idea of the SDM is to aid the designer to cope with the complexity involved in system design; an unpleasant interface is not consistent with this goal.

8.5 Linkage to the Detailed Design Stage

Another area for further research concerns the question of linking a SDM-based design problem structure to the system detailed design process. This problem is regarded by Huff as the single most important problem to be addressed in future SDM-related research (see [Section 9.1, Huff June(b) 1979]).

Basically, the SDM design framework is seen as a structuring scheme for thinking about alternative implementation techniques and deciding upon the most appropriate ones in an organized fashion. Based on the experience learned from the current application study, it seems that SDM can be applied continuously, in various ways, to aid the organization of subsequent design activities. For example, as a result of carefully studying the trade-offs and concurrencies in the design framework, the original graph model may be modified in terms of the existence and strength of the interdependencies. The modified graph model can then be decomposed so as to obtain a decomposition possibly different from the previous design problem structure. The difference is due to the fact that while the previous design problem structure is based on various implementation alternatives, the new decomposition is based on implementation approaches that have been decided upon.

Each subproblem in the newly obtained decomposition can then be attacked individually (each subproblem may be assigned to a design team). Given each subproblem as a subsystem, SDM can be applied to the subsystem in a similar manner as before: more detailed requirements can be generated and interdependency assessment activity will be performed to define an SDM graph model which in turn will be decomposed. By applying SDM in this recursive fashion, it is conceivable that at some point implementation issues one thinks about for performing interdependency assessment would heavily concern the alternative ways for the design of software modules. This is probably a reasonable entry point to the "detailed design stage."

Of course, whether the above suggestion is the right approach to the linkage problem or not requires further study and testing.

AREAS FOR FURTHER RESEARCH AND IMPROVEMENT

8.6 Summary

While still a research project, the Systematic Design Methodology is proving its effectiveness in aiding system architects to organize and manage the many and diverse requirements typical of complex systems design. While again confirming its fundamental soundness and value, this study has suggested new improvements that may be made to the methodology and has pointed out areas that deserve further research.

The faith which the designers and the author (who have participated the current application study) have in the potential of the methodolog to improve quality of complex system design is reflected in our being willing to try SDM on future projects. Hopefully, this study will motivate further development and testing work on the methodology.

REFERENCES

Alexander, C.: Notes on the Synthesis of Form, Harvard University Press, 1964.

Anderson, R.: Handbook of Logic Circuit Testing, Omnicomp Inc., Phoenix, AZ 1975.

Andreu, R.C. and Madnick, S.E.: "An Excercise in Software Architectural Design: From Requiremets to Design Problem Structure", Technical Report no. 3, Sloan School of Management, MIT, November 1977.

Andreu, R.C. and Madnick, S.E.: "Completing the Requirements Set as a Means Towards Better Design Frameworks: A Follow-up Excercise in Software Architectural Design", Technical Report no. 4, Sloan School of Management, MIT, December 1977.

Andreu, R. C.: "A Systematic Approach to the Design and Structuing of Complex Software Systems", PhD. Thesis, Sloan School of Management, M.I.T., Cambridge, Mass. 1978.

Armstrong, J. R., Woodruff, G., "Simulation Techniques for Microprocessors", Proceeding of Design Automation Conference No.14,, June 1977.

Beldford, P., et. al.: "Specifications: The Key to Effective Software Development", Proc. 2nd. Int. Conf. on Soft. Eng., 1976.

Breuer, M.A., Friedman A.D.: Diagnosis and Reliable Design of Digital System, Computer Science Press, Woodland Hills, CA 1976.

DeWolf, B.: "Requirements Specification and Design for Real-time System : A Problem Statement", IR&D Memo No. 4, C.S. Draper Labs., Jan. 1977.

Endres, A.B.: "An Analysis of Errors and Their Causes in System Programs", IEEE Transactions on Software Engineering, June, 1975.

Fagan, M.E.: "Design and Code Inspection and Process Control in the Development of Programs", IBM TR 21-572, December, 1974.

Freeman, P.: "The Context of Design", in "Tutorial on Software Design Techniques", P. Freeman and A.I. Wasserman, Eds. - IEEE Catalog No. 76CH1145-2C.

REFERENCES

Grason, J., "Testing Circuit Packs Containing LSI components", ELECTRO-77, Session 32, "Testing Complex Digital Assemblies".

Holden, T.: "A Systematic Approach to Designing Complex System: Application to Software Operating Systems", Technical no. 5, Sloan School of Management, MIT, May 1978.

Howland, J. C.: "An Examination of Systems Integration", TSP Division, GenRad Inc., August 1978.

Huff, S. L., and Madnick, S. E.: "An Approach to the Construction of Functional Requirement Statements for System Architectural Design", Technical Report no. 6, Sloan School of Management, MIT, June 1978.

Huff, S. L., and Madnick, S. E.: "An Extended Model for a Systematic Approach to the Design of Complex Systems", Technical Report no. 7, Sloan School of Management, MIT, July 1978.

Huff, S. L.: "A Methodology for Designing the Architecture of Complex Systems", Doctoral Thesis Proposal, Sloan School of Management, MIT, Oct. 1978.

Huff, S. L., and Madnick, S. E.: "Decomposition of Weighted Graphs Using the Interchange Partitioning Technique", Technical Report no. 8, Sloan School of Management MIT, Jan. 1979.

Huff, S. L., and Madnick, S. E.: "Analysis Techniques for Use With The Extended SDM Model", Technical Report no. 9, Sloan School of Management MIT, Feb. 1979.

Huff, S.L.: "Architectural Design of a New M.I.T. Budgeting System: An Application of the Systematic Design Methodology", Technical Report no. 11, Sloan School of Management MIT, June(a) 1979.

Huff, S.L.: "A Systematic Methodology for Designing the Architecture of Complex Software Systems", PhD. Thesis, Sloan School of Management, M.I.T., Cambridge, Mass., June(b) 1979.

Madnick, S. E., and J. Donovan: Operating Systems, McGraw-Hill 1975.

Myers, G.: Structured/Composite Design, Van Nostrand Reinhold Co., 1978.

REFERENCES

Simon, H.: The Science of the Artificial, MIT Press, 1969.

Stevens, J., et. al.: "Structured Design", IBM System Journal, Vol. 13, no. 2, April 1974.

Szygenda, S.A.: "Digital System Simulation", Computer, March 1975.

Thayer, T.A.: "Understanding Software through Analysis of Empirical Data", Procs., 1975 National Computer Conference -AFIPS.

Wasserman, T., et. al.: "Software Engineering: The Turning Point", Computer, September 1978.

White, J., and T. Booth: "Towards an Engineering Approach to Software Design", Proc. 2nd Int. Conf. on Soft. Eng., 1976.

APPENDIX A

FINAL SET OF REQUIREMENTS FOR THE TEST PROGRAM PREPARATION SYSTEM AS USED IN SDM ANALYSIS(*)

[1] There will be a test language in which user can write test procedures to control the testing.

COMMENT --

Digital test commands will be specified in the new test language format, which is very similar to the current digital commands.
The non-digital commands can be specified in current format.
Errors must be properly reported so as to enable the user to quickly diagnose and locate the error.
Furthermore, errors can be optionally corrected on-line as in the current system.
This requires an automatic chaining into the editor upon errors and back into the processor for

---

(*) Each functional statement may have a Comment Section attached to it. Information that the designers desire to have formally stated in the specifications but was deemed not appropriate to be specificied as part of the SDM requirement statement was included in the Comment Section.

retries.

The original source file must be replaced with the output listing file.

[2] There will be a test program processing facility.

COMMENT --

For non-digital commands, which are not to be used by the simulator, there will be syntax error checking on them within test program processing. No binary translation will be done. At the same time, we may want to generate the TP Label Table.

[3] There will be a test program macro processing facility.

[4] There will be a test program macro library.

COMMENT --

Macros can be inserted into or deleted from the library.

APPENDIX A

There may be syntax error checking before a macro definition is inserted into library.

[5] Default driving characteristics associated with output or bidirectional pins for each standard device type will be supported.

[6] Default driving characteristics associated with output or bidirectional pins for each standard device type can be overriden by user.

[7] Incremental test program compilation will be supported when more test steps are added to the end of the test program.

[8] Simulation results for a device model can be overriden by user.

COMMENT --

This can be done using transitionals.
Specific transitionals can be turned on or off within test program.
There can be multiple transitional changes within

the same test step.

Support to transitionals will be consistent with characteristics of the simulator:

Transitionals can have H,L,X,Z value.,

Different delays can be assigned to multiple transitional changes within the same test step.

The timing and control of occurrence of the transitional change must be with reference to driver timing plus a delay.

Transitionals attached to output pins will take on the output drive characteristics of the IC pin.

Transitionals attached to input pins will change only that pin without affecting the nodal value.

[9] Output of a device for which no simulation model exists can be controlled by user.

COMMENT --

Use transitionals to support this.

[10] The GO/NOGO test program will be generated.

COMMENT --

The good board simulator will generate the output patterns, based on the translated test program database and circuit description.
Results of good board simulation, i.e., the output patterns and the original test program will be merged together to produce the GO/NOGO test program.
Non-digital test commands originally specified in the user-created test program, which are not used for simulation, should also appear in the GO/NO-GO test program.

[11] There will be New-to-Current GO/NO-GO test program conversion facilities.

COMMENT --

The conversion here involves translating any test commands whose new format is different from the current format.
GO/NOGO program can be manually generated by the user.
Macro processing facility is not supported in

this case.

[12] A test program can contain multiple sets of input test patterns each testing a different part of the same board.

[13] A test program can contain multiple sets of input test patterns all of which test the same part of a board.

[14] There will be Current-to-New test program conversion facilities.

COMMENT --

This does not have to be interactive or fast enough to allow continued ongoing use of the old digital test language.

[15] There will be a Circuit Description Language which allows the user to specify board-related or IC-related descriptions.

COMMENT --

APPENDIX A

Circuit description will contain schematic diagram of the UUT:

The UUT schematic diagram description will contain the list of components on the circuit board, their interconnections, and their connections to the external signal paths.

Circuit description source will contain Tester-UUT interface description.

Circuit description source can also contain user-specified attributes, for example like NONAIL, etc.

Each external signal path, IC, and special device will be assigned an unique name. Devices must be allowed to be mentioned more than once.

Each physical package is considered a single component and therefore will be assigned a single name.

Relevant attributes supported by current test system will be supported.

[16] Format of Circuit Description Source file will be compatible with that of In-Circuit test systems.

APPENDIX A

COMMENT --

This implies that Circuit Description Source will be divided into sections preceded by a section header (%CIRUICT, ADAPTOR, etc.).

[17] Different sections of the circuit description source file can be used by more than one product type if so specified.

COMMENT --

For example, %CIRCUIT : A,B; specifies that the %CIRCUIT section is valid for both tester type A and tester type B.

[18] Different sections of the circuit description source can be shared by multiple sets of input patterns in the same test program which have the identical circuit topology.

[19] There will be a circuit description processing facility.

APPENDIX A

COMMENT --

There will be error checking during circuit description processing. The most common errors are errors in the connection field in the %circuit section.
The best solution is a solution that minimizes the processing between the start of processing and the first error report. Not acceptable is a solution that processes most or all of the Circuit Description File before reporting any errors.

[20] There will be partial re-processing facility for circuit description source file.

[21] Default driving characteristics of output or bidirectional pins for each specific tester will be supported.

COMMENT --

For each output or bidirectional pin, there will

be overdrive, hard, and soft associated with its "0" and "1" states.
The default information must be kept somewhere in the system. CDL must allow user to indicate the tester type.

[22] Default driving characteristics of output or bidirectional pins for each specific tester can be overriden by user.

COMMENT --

For each specific tester, user may override the default in %ADAPTOR.
CDL must allow user to override the default.

[23] Board-related statistical data will be reported to user.

[24] Board-related warnings will be reported to user.

[25] Circuit network connection cross-reference data will be reported to user.

APPENDIX A

[26] There will be Current-to-New circuit description source file conversion facilities.

COMMENT --

The translator must be able to process source imbedded with line numbers and error messages.

[27] There will IC model libraries.

COMMENT --

At the minimum, there will be a system library, which must be locked, a system update library, and one or more user libraries.

[28] About 90% of the existing System IC model library will be converted.

[29] Every widely used primitive element and SSP Boot element will be supported.

[30] Models for ROM's will be supported.

APPENDIX A

COMMENT --

Alternative techniques for ROM space saving will be implemented.
%ROM section may be used, in Circuit Description Source File, to contain the content of a specific ROM.

[31] Default transport delay characteristics associated with output pins of each standard IC family type will be supported.

[32] There will be IC library maintenance facilities.

COMMENT --

Content of the library can be obtained.
IC can be inserted into the library.
IC can be deleted from the library.
A binary copy of a model can be extracted from one library and then inserted into another library.

[33] There will be an IC model information display

facility.

COMMENT --

The information displayed shall be that information that is easy to obtain and display.

[34] IC's can be modeled using circuit description technique.

COMMENT --

%LIB: replaces the $ processing for IC model circuit descriptions. The format of the section follows that for the current system with additions to support the universal bus device and transport delays.

[35] Models for Gate Arrays (PLA devices) will be supported.

COMMENT --

Gate Arrays and PLA refer to the same category of

devices. There will be an upper limit on the size of PLA device, for now, PLA devices as large as 2000 gates or smaller will be modeled.

[36] Default transport delay characteristics associated with output pins of each standard IC family type can be overriden by user on a pin by pin base.

[37] IC's can be modeled using functional modeling technique (SCL-II).

COMMENT --

This requirement calls for the language (Simulation Command Language, Version II) itself and the language processor (be it a compiler or interpreter).

[38] SCL-II will be convenient to use for both system manufacturer and customer.

COMMENT --

Machinary of the simulator will be hidden from SCL-II users.

[39] SCL-II will be efficient enough to be suitable for some in-house modeling of primitives.

COMMENT --

Both speed efficiency and core efficiency.

[40] There will be SCL-II debugging tools.

COMMENT --

SCL-II debugging will be supported by the good board simulator to make SCL-II a good tool for the average IC modeler.
A clean interface between SCL-II models and the simulator should be maintained for debugging purpose.
Don't forget to debug fault models.

[41] There will be tools aiding .SR-type IC models conversion.

APPENDIX A

[42] There will be tools aiding SCL-I TO SCL-II IC models conversion.

[43] There will be a conditional connectivity language for user to define connectivity of an IC model.

COMMENT --

For both .SR-models and SCL-II models.

[44] Connectivity of .SR-IC models can be automatically simulated using connectivity simulator.

[45] There will be a (good board) simulator.

COMMENT -

The simulator will be used for both good value simulation and fault simulation.
The simulator will be a multi-delay simulator.
The simulator will be at least a 4-value simulator (4-value: 0,1,X,Z).

APPENDIX A

[46] There will be fault simulation activity.

COMMENT -

This requirement implys all the work needed to be done prior to simulation and fault simulation itself.
Work needed to be done prior to simulation includes fault analysis, fault insertion, perhaps fault collapsing (for speed reason), etc.
Fault simulation itself uses the good board simulator.

[47] The simulator will be as fast as possible.

COMMENT --

The new simulator will be at least 5-10 times as fast as the current simulator.
The simulator will be a selective-trace simulator for the purpose of faster speed.

[48] Results of good board simulation can be properly displayed.

[49] Incremental simulation will be supported when more steps are added at the end of the test program.

COMMENT --

This applies to both good board and fault simulation.

[50] Output pin transport delay characteristics will be taken into account by the simulator.

COMMENT --

The purpose of having a multi-delay simulator is to satisfy this requirement.

[51] Default transport delay characteristics associated with output pins of each specific tester type will be supported.

[52] Default output pin transport delay characteristics

for each tester type can be overriden by user.

[53] Bus type logic will be properly handled.

COMMENT --

The simulator and the circuit description assembler must be smart about bus connections, the new Z state and family types. Any output or bidirectional pin will have overdrive, hard, and soft associated with its "0" and "1" states.
Whenever pins, including drivers, are bused together, the appropriate information will be loaded into the universal bus device that combines bus output values.

[54] Properties of a particular target tester will be taken into account by the simulator.

COMMENT --

For example, driver pins characteristics, 12*12 feature, etc.. (Not include synchronization)

APPENDIX A

[55] Tester will be better synchronized to the simulator.

COMMENT --

Some effort should be made to synchronize the tester to the simulator just as it is now possible to synchronize the tester to the board.

[56] Fault simulation will be as fast as possible.

COMMENT --

Fault simulation speed will be about 10-20 times faster.
A fast good board simulator will serve as the base for fast fault
simulation. Other techniques can be used to speed up fault simulation, for example, fault activation sorting and late start-up, fault collapsing.

[57] Test program fault coverage scoring will be reported to user.

[58] Detected/Undetected faults will be reported to user.

[59] Internal node faulty activity for an undetected fault can be displayed to user.

[60] Aborted fault simulation can be resumed.

[61] Faults to be simulated can be selected by user.

[62] Fault simulation can be resumed for more resolution.

COMMENT --

Normally, simulation stops at the first step where the fault is detected.
This requirement allows more than one fault signature be simulated for a single fault.

[63] More than one fault signature can be stored for a single fault.

COMMENT --

This feature can be used for X-detects or poor

signature resolution.

[64] The Stuck type fault models will be supported.

COMMENT --

Including:

Output-stuck-at-1 fault model

Output-stuck-at-0 fault model

Input-stuck-at-0 fault model

Input-stuck-at-1 fault model

An output pin stuck-at fault model must comply with the drive characteristics of the stuck pin.

[65] There can be user-inserted faults.

[66] User-inserted faults will be consistent with fault insertion machinary.

[67] Some form of internal LSI fault models will be supported.

[68] Default transport delay characteristics associated

with output pins of each standard IC family type will be supported.

[69] Default transport delay characteristics associated with output pins of each standard IC family type can be overriden by user on a pin by pin base.

APPENDIX B

INTERDEPENDENCY ASSESSMENT STATEMENTS FOR
THE TEST PROGRAM PREPARATION SYSTEM

FROM	TO	WEIGHT	DESCRIPTION
[1]	[2]	S	THE PROCESSOR MUST PROPERLY PROCESS THE TEST PROGRAM.
[1]	[3]	W	MACRO PROCESSING FACILITY MAY HAVE TO KNOW ABOUT THE TEST LANGUAGE.
[1]	[8]	W	TEST LANGUAGE MUST INCLUDE CONSTRUCTS TO SUPPORT TRANSITIONALS.
[1]	[9]	W	SEE [1], [8].
[1]	[10]	A	GO/NO-GO GENERATOR HAS TO UNDERSTAND THE LANGUAGE.
[1]	[11]	A	THE TRANSLATOR MUST UNDERSTAND THE TEST LANGUAGE.
[1]	[12]	W	TEST LANUGUAGE MUST SUPPORT THE SPECIFICATION OF EACH SECTION.
[1]	[13]	W	SAME AS [1], [12].
[1]	[14]	A	THE TRANSLATOR MUST UNDERSTAND THE LANGUAGE.
[1]	[15]	W	EXTERNAL NAMES ARE PASSED TO TEST PROGRAM.
[1]	[45]	W	DELAY, VALUES, ETC., CAN BE SPECIFIED IN TEST LANGUAGE.
[1]	[53]	W	USE TEST LANGUAGE TO SPECIFY BUS DIRECTIONS.
[1]	[55]	W	SYNCHRONIZATION WILL BE SPECIFIED IN TEST LANGUAGE.
[2]	[7]	A	SAVE INTERNAL VALUES FOR INCREMENTAL COMPILATION.
[2]	[8]	W	PROPERLY PROCESS TRANSITIONALS.

APPENDIX B

[2]	[9]	W	SAME AS [2],[8].
[2]	[10]	A	TP PROCESSOR MAY BE USED TO GENERATE GO/NO-GO.
[2]	[12]	W	TRANSLATOR MUST PROPERLY TRANSLATE "MULTIPLE SECTIONS."
[2]	[13]	W	SAME AS [2], [12].
[2]	[19]	W	EXTERNAL NAMES HAVE TO BE PASSED TO TP TRANSLATOR.
[2]	[45]	W	INFORMATION GENERATED BY THE TP TRANSLATOR IS NEEDED FOR SIMULATION.
[2]	[46]	W	INFORMATION GENERATED BY THE TP TRANSLATOR IS NEEDED FOR FAULT ANALYSIS.
[2]	[48]	W	CORRESPONDENCE BETWEEN "LABELED TEST STEP" AND "TEST STEP" IS NEEDED.
[2]	[53]	W	BUS DIRECTION INFORMATION MUST BE PASSED TO SIMULATOR FROM TP TRANSLATOR.
[2]	[55]	W	SYNCHRONIZATION INFORMATION MUST BE PASSED TO THE SIMULATOR.
[3]	[4]	A	MACRO PROCESSOR MAY USE MACRO LIBRARY; LIBRARY WILL BE CREATED AND MODIFIED BY THE PROCESSOR.
[5]	[15]	W	USE CDL TO INDICATE IC FAMILY TYPE.
[5]	[19]	W	THE PRE-PROCESSOR MAY PROVIDE THE SIMULATOR WITH THE DEFAULT INFORMATION.
[6]	[15]	A	CDL WILL SUPPORT USER TO OVERRIDE THE DEFAULT.
[6]	[19]	A	THE PRE-PROCESSOR MAY PROVIDE THE SIMULATOR WITH THE OVERRIDEN INFORMATION.
[7]	[10]	A	GO/NO-GO MAY BE GENERATED

			INCREMENTALLY.
[7]	[49]	W	PROBABLY INTERACT BECAUSE BOTH SUPPORT INCREMENTAL PROGRAM PREPARATION.
[8]	[9]	S	[9] IS SUPPORTED BY [8].
[8]	[15]	W	THERE MUST BE A WAY FOR THE TEST LANGUAGE COMMAND TO REFER TO AN IC PIN.
[8]	[19]	A	PRE-PROCESSOR MUST PROPERLY HANDLE TRANSITIONALS TO SUPPORT THE SIMULATOR.
[8]	[45]	W	SUPPORT TO TRANSITIONALS WILL BE CONSISTENT WITH THE CHARACTERISTICS OF THE SIMULATOR.
[9]	[15]	A	CDL MUST PROVIDE CONSTRUCTS TO HANDLE DEVICES THAT CANNOT BE MODELED.
[9]	[19]	A	SAME AS [8],[19].
[9]	[45]	W	SAME AS [9], [45].
[10]	[12]	W	GO/NO-GO GENERATOR MUST MERGE THE SIMULATION RESULTS INTO THE RIGHT SECTION.
[10]	[13]	W	SAME AS [10], [12].
[10]	[45]	W	GOOD VALUES ARE NEEDED FOR GO/NO-GO TEST.
[10]	[49]	W	MUST BE ABLE TO TELL WHICH GO/NO-GO ARE ACCEPTED OR REJECTED..
[10]	[55]	W	TESTER-SIMULATOR SYNCHRONIZATION HAS TO BE TRANSLATED TO TESTER-BOARD SYNCHRONIZATION.
[12]	[13]	A	BOTH NEED TO USE MULTIPLE SECTIONS IN THE TEST PROGRAM.
[15]	[16]	W	CDL HAS TO SUPPORT SPECIFICATION OF

			THE SECTIONS.
[15]	[17]	W	CDL HAS TO SUPPORT SPECIFICATION OF SECTIONS.
[15]	[18]	W	SAME AS [15] [17].
[15]	[19]	S	[19] PROCESSES [15].
[15]	[20]	S	SAME AS [15], [19].
[15]	[21]	W	USE CDL TO SPECIFY THE SPECIFIC TESTER OR IC TYPE.
[15]	[22]	A	USE CDL TO SPECIFY THE OVERRIDEN INFORMATION.
[15]	[26]	S	TRANSLATOR MUST UNDERSTAND SYNTAX OF THE NEW CDL.
[15]	[30]	W	MAY USE A %ROM SECTION FOR PROVIDING CONTENTS OF A ROM MODEL.
[15]	[31]	W	USE CDL TO INDICATE THE IC FAMILY TYPE.
[15]	[34]	S	.SR-IC'S ARE MODELED USING CDL.
[15]	[35]	W	MAY USE %PLA FOR SPECIFYING CONTENTS OF A PLA.
[15]	[36]	A	USE CDL TO OVERRIDE DEFAULT.
[15]	[43]	A	CCL MAY BE A SUBSET OF CDL.
[15]	[51]	W	USE CDL TO INDICATE WHICH SPECIFIC TESTER OR IC TYPE.
[15]	[52]	A	USE CDL TO OVERRIDE THE DEFAULT.
[15]	[55]	W	CDL MAY BE USED TO TELL THE SIMULATOR HOW TO SYNCHRONIZE WITH THE TESTER.
[15]	[68]	W	USE CDL TO INDICATE IC FAMILY TYPE.
[15]	[69]	A	USE CDL TO OVERRIDE THE DEFAULT.

APPENDIX B

[16]	[19]	W	PROCESSOR PROCESSES SOURCE WHICH IS IN THE "SECTIONS" FORM.
[16]	[20]	W	SEE [16], [19].
[16]	[26]	A	CURRENT-TO-NEW TRANSLATOR MUST TRANSLATE CURRENT SOURCES INTO "SECTIONS" FORM.
[16]	[34]	A	USE %LIB AS ONE SECTION OF THE SOURCE.
[16]	[43]	W	CONDITIONAL CONNECTIVITY MAY BE ONE SECTION.
[17]	[19]	W	PROCESSOR MUST RECOGNIZE THE PROPER SECTIONS.
[17]	[20]	W	SAME AS [17], [19].
[18]	[19]	A	PRE-PROCESSOR MUST PROPERLY PREPARE THE DATA BASE FOR THE MULTIPLE SECTIONS WHICH HAVE THE SAME CIRCUIT TOPOLOGY IN ORDER TO REDUCE DATA REDUNDANCY.
[18]	[20]	W	SEE [18], [19].
[19]	[20]	A	REQUIRE COMMON PROCESSING.
[19]	[21]	W	PRE-PROCESSOR SHOULD PROVIDE THE SIMULATOR WITH THE DEFAULT INFORMATION.
[19]	[22]	W	PRE-PROCESSOR SHOULD PROVIDE THE SIMULATOR WITH THE OVERRIDEN INFORMATION.
[19]	[23]	A	REPORT GENERATION USES DATA BASE PRODUCED BY THE PROCESSOR.
[19]	[24]	A	SAME AS [19], [23].
[19]	[25]	A	SAME AS [19], [23].
[19]	[27]	W	PRE-PROCESSOR NEEDS INFORMATION FROM THE LIBRARY AND THUS MAY NEED TO KNOW HOW THE DATA IN THE LIBRARY ARE

APPENDIX B

ORGANIZ ..

[19]	[31]	W	PRE-PROCESSOR HAS TO PROVIDE SIMULATOR WITH THE DEFAULT.
[19]	[32]	W	MAY USE COMMON MECHANISM TO GET ACCESS TO THE LIBRARY.
[19]	[33]	W	SAME AS [19], [32].
[19]	[34]	S	PRE-PROCESSOR HANDLES BOTH CIRCUIT BOARDS AND IC MODELS.
[19]	[45]	A	PRE-PROCESSOR GENERATES INFORMATION NEEDED BY THE SIMULATOR.
[19]	[46]	A	PRE-PROCESSOR GENERATES INFORMATION NEEDED BY FAULT ANALYSIS.
[19]	[51]	W	PRE-PROCESSOR SHOULD PROVIDE SIMULATOR WITH THE DEFAULT.
[19]	[52]	A	PRE-PROCESSOR CAN PROVIDE THE SIMULATOR WITH THE OVERRIDEN INFORMATION.
[19]	[53]	A	PRE-PROCESSOR IS RESPONSIBLE FOR BUILDING THE UNIVERSAL BUS DEVICE.
[19]	[68]	W	PRE-PROCESSOR WILL PROVIDE THE SIMULATOR WITH THE DEFAULT.
[19]	[69]	A	PRE-PROCESSOR CAN PROVIDE THE SIMULATOR WITH THE OVERRIDEN INFORMATION.
[20]	[23]	A	REPORT MAY BE RE-GENERATED AS A RESULT OF RE-PROCESSING.
[20]	[24]	A	SAME AS [20], [23].
[20]	[25]	A	SAME AS [20], [23].
[20]	[27]	W	INFORMATION ABOUT A PARTICULAR ELEMENT IN THE LIBRARY CAN OR CANNOT BE CHANGED AS A RESULT OF PARTIAL RE-PROCESSING.

APPENDIX B

[20]	[45]	W	PARTIAL RE-PROCESSING MAY UPDATE INFORMATION NEEDED BY SIMULATOR.
[20]	[46]	W	PARTIAL RE-PROCESSING MAY UPDATE INFORMATION NEEDED BY FAULT ANALYSIS, ETC..
[23]	[24]	A	BOTH CAN BE GENERATED FROM THE SAME DATA BASE, THUS CAN BE GENERATED SIMULTANEOUSLY.
[23]	[25]	A	SAME AS [23], [24].
[24]	[25]	A	SAME AS [24], [25].
[26]	[41]	S	CONVERSION OF CIRCUIT DESCRIPTION FILE FOR IC MODELS CORRESPONDS TO .SR-IC TYPE MODEL CONVERSION.
[27]	[30]	A	ROM MODEL IS ONE TYPE OF IC MODEL.
[27]	[32]	A	MAY USE COMMON ACCESS TOOLS.
[27]	[33]	A	MAY USE COMMON ACCESS TOOLS.
[27]	[34]	A	ONE TYPE OF IC MODEL.
[27]	[35]	A	ONE TYPE OF IC MODEL.
[27]	[37]	A	ONE TYPE OF IC MODEL.
[28]	[29]	S	[29] GREATLY EASES THE CONVERSION.
[28]	[41]	A	.SR MODELS CONVERSION IS PART OF THE LIBRARY CONVERSION.
[28]	[42]	A	SCL-I MODELS CONVERSION IS PART OF THE LIBRARY CONVERSION.
[29]	[41]	A	[29] EASES .SR-IC MODELS CONVERSION.
[29]	[42]	A	SEE [29], [41].
[30]	[35]	W	MAY SHARE COMMON TOOLS.
[32]	[33]	W	BOTH MAY USE THE SAME MECHANISM TO GET ACCESS TO THE ELEMENTS OF THE LIBRARY.

[33]	[44]	A	CONNECTIVITY SIMULATION IS PART OF THE .SR-IC MODEL GENERATION.
[34]	[45]	W	THE SIMULATOR HAS TO KNOW WHICH NODES ARE EXTERNAL NODES.
[34]	[53]	W	.SR-IC MODELS MUST SUPPORT PROPER HANDLING OF BUS TYPE DEVICES.
[34]	[55]	W	CONSIDER PIN ATTRIBUTE "CLOCK".
[34]	[67]	A	[34] WILL SUPPORT [67].
[37]	[38]	W	[38]-[40] SCL-II RELATED.
[37]	[39]	A	SEE 37,38.
[37]	[40]	A	SEE 37,38.
[37]	[42]	A	THE CONVERSION TOOL MUST UNDERSTAND SCL-II; DESIGN OF SCL-II MUST TAKE THIS CONVERSION REQUIREMENT INTO CONSIDERATION.
[37]	[45]	A	SCL-II INTERPRETER IS PART OF THE SIMULATOR.
[37]	[47]	W	SEE [37], [45].
[37]	[53]	W	SEE [34], [53].
[37]	[55]	W	SEE [34], [55].
[37]	[67]	A	SEE [34],[67].
[38]	[39]	W	MAY CONFLICT WITH EACH OTHER.
[38]	[40]	W	DEBUGGING IS A "CONVENIENCE ISSUE."
[38]	[45]	A	[38] IMPLYS SUPPORT NEEDED FROM THE SIMULATOR.
[38]	[46]	W	BEING CONVENIENT TO USE SCL-II HELPS FAULT INSERTION (SCL-II SUPPORTS FAULT INSERTION).
[38]	[65]	A	"CONVENIENT TO USE SCL-II" ALSO

			MEANS "CONVENIENT FOR FAULT MODELING".
[38]	[67]	W	SCL-II SUPPORTS INTERNAL LSI FAULT MODELS.
[39]	[47]	A	EFFICIENCY OF SCL-II AFFECTS SIMULATOR'S SPEED.
[40]	[45]	A	SCL-II DEBUGGING MUST BE SUPPORTED BY THE SIMULATOR.
[40]	[46]	W	SCL-I DEBUGGING MUST BE SUPPORTED BY FAULT SIMULATION.
[40]	[49]	W	INCREMENTAL SIMULATION COMPLICATES SCL-II DEBUGGING.
[40]	[55]	A	[55] COMPLICATES [40].
[40]	[59]	W	INTERNAL FAULTY ACTIVITY RECORDS HELP DEBUGGING.
[40]	[67]	W	SEE [40], [62].
[44]	[45]	S	USE THE SAME SIMULATOR.
[44]	[50]	W	MAKE SURE THAT OUTPUT PIN CHARACTERISTICS ARE TAKEN INTO ACCOUNT.
[44]	[53]	W	MAKE SURE BUS TYPE LOGIC IS PROPERLY HANDLED.
[45]	[46]	A	FAULT SIMULATION IS MAINLY BASED ON GOOD BOARD SIMULATOR.
[45]	[47]	S	SPEED CONSIDERATION FOR GOOD BOARD SIMULATION.
[45]	[48]	S	DISPLAY OF GOOD BOARD SIMULATION RESULTS.
[45]	[49]	W	INCREMENTAL SIMULATION MAY ADD BURDEN TO THE SIMULATOR.
[45]	[50]	A	PUTS BURDEN ON SIMULATOR: THE SIMULTOR WILL BE A MULTI-DELAY

			SIMULATOR IN ORDER TO SATISFY [50].
[45]	[53]	W	[53] PUTS BURDEN ON SIMULATOR: SOME MACHINARY MUST BE NEEDED BY THE SIMULATOR TO USE UNIVERSAL BUS DEVICE.
[45]	[54]	W	[54] ADDS BURDEN ON SIMULATOR.
[45]	[55]	A	[55] PUTS BURDEN ON SIMULATOR.
[46]	[49]	W	CAPABLE OF DOING INCREMENTAL FAULT SIMULATION.
[46]	[55]	W	FAULT SIMULATION HAS TO TAKE SYNCHRONIZATION INTO CONSIDERATION.
[46]	[56]	A	SPEED CONSIDERATION OF FAULT SIMULATION.
[46]	[57]	W	TEST PROGRAM FAULT COVERAGE SCORING IS OBTAINED AS A RESULT OF FAULT SIMULATION.
[46]	[58]	A	SEE [46], [57].
[46]	[59]	A	DISPLAY IS AFFECTED BY HOW FAULT SIMULATION IS CARRIED OUT.
[46]	[60]	W	[60] MAY COMPLICATE FAULT SIMULATION.
[46]	[61]	W	FAULT ANALYSIS REQUIRED FOR FAULT SELECTION.
[46]	[62]	W	[62] COMPLICATES FAULT SIMULATION.
[46]	[63]	W	[63] COMPLICATES FAULT SIMULATION.
[46]	[64]	S	FAULT ANALYSIS IS REQUIRED FOR EACH FAULT MODEL.
[46]	[65]	S	SEE [46],[64].
[46]	[66]	A	[46] DEFINES THE MACHINARY.
[46]	[67]	A	[46] SUPPORTS [67].

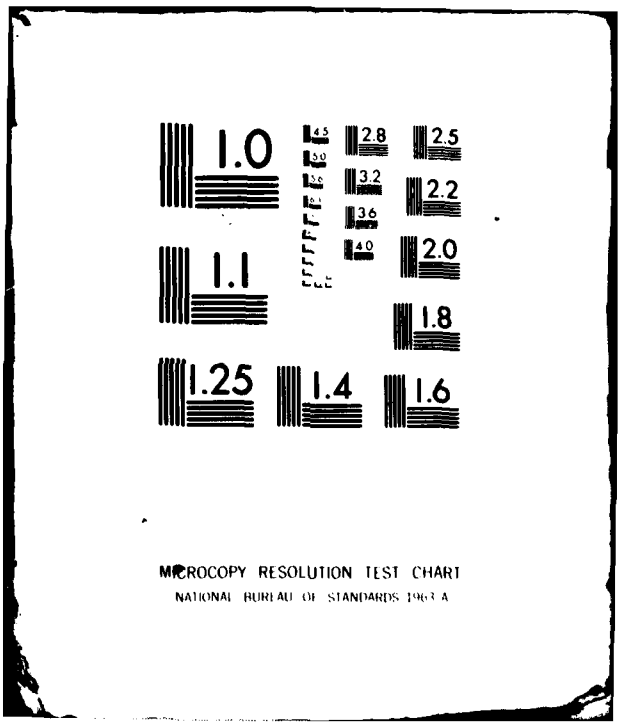
APPENDIX B

[48]	[49]	W	SIMULATOR MUST RECORD OUTPUTS MUST INCREMENTALLY.
[48]	[55]	W	SYNCHRONIZATION/TIMING WILL HAVE TO BE DISPLAYED.
[48]	[59]	A	MAY USE SIMILAR MECHANISM.
[49]	[55]	W	SYNCHRONIZATION ACROSS INCREMENTS MUST BE HANDLED.
[49]	[57]	W	INCREMENTAL FAULT SIMULATION AFFECTS 57-63 LOGISTICALLY (DUE TO THE FACT INCREMENTAL PROCESS MAY BE REJECTED).
[49]	[58]	W	SEE [49], [57].
[49]	[59]	W	SEE [49], [57].
[49]	[60]	W	SEE [49], [57].
[49]	[61]	W	SEE [49], [57].
[49]	[63]	W	SEE [49], [57].
[50]	[54]	A	[50] IS A MAJOR PROPERTY OF A SPECIFIC TESTER.
[53]	[54]	A	[53] IS A MAJOR PROPERTY OF A SPECIFIC TESTER.
[53]	[64]	W	[53] COMPLICATES [64].
[54]	[64]	W	DEPENDING ON THE OUTPUT PIN CHARACTERISTICS OF THE D/S'S OF A PARTICULAR TESTER, USE THE PROPER TYPE OF FAULT MODEL.
[54]	[65]	W	SAME AS [54], [64].
[55]	[60]	W	[55] ADDS BURDEN ON [60].
[55]	[62]	W	SEE [55], [60].
[55]	[64]	W	FAULTS WILL CAUSE SYNCHRONIZATION BE INTERRUPTED.

APPENDIX B

[55]	[65]	W	SEE [55], [64]
[56]	[57]	W	[57] SLOWS DOWN FAULT SIMULATION SPEED.
[56]	[65]	W	SAME AS ABOVE.
[56]	[67]	W	SAME AS ABOVE.
[57]	[60]	W	[60] COMPLICATES [57].
[57]	[61]	W	SEE [57], [60].
[57]	[63]	A	MAY HAVE TO REPORT CASES WHERE MORE THAN 1 FAULT RESOLUTION ARE OBTAINED.
[59]	[61]	W	MAY USE THE SAME "FAULT SELECTION" MECHANISM.
[60]	[61]	W	[61] COMPLICATES [60]: ANOTHER PIECE OF INFORMATION TO STORE.
[60]	[62]	W	SAME AS [60] [61].
[60]	[63]	W	SAME AS [60] [61].
[61]	[62]	W	COMMON SELECTION MECHANISM.
[62]	[63]	S	[62] IS NEEDED FOR [63].
[64]	[65]	S	[65] COMPLICATES [64].
[64]	[66]	S	[66] EASES [64].
[64]	[67]	S	[67] COMPLICATES [64].
[65]	[66]	S	[66] EASES [65]'S JOB.
[65]	[67]	A	THERE CAN BE LSI INTERNAL FAULTS INSERTED BY USER.
[66]	[67]	W	LSI INTERNAL FAULT MODELS CAN BE A SUBSET OF USER INSERTED FAULTS.

AD-A090 707 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
A SYSTEMATIC APPROACH TO COMPLEX SYSTEM DESIGN: AN APPLICATION --ETC(U)
MAY 80 P TUNG N00039-80-K-0573
UNCLASSIFIED CISR-P010-80002-13 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

APPENDIX C

REQUIREMENTS SUBSETS DERIVED FROM THE BEST SYSTEM DECOMPOSITION

SUBPROBLEM 1 -- TEST PROGRAM PROCESSING

[1] There will be a test language in which user can write test procedures to control the testing.

[2] There will be a test program processing facility.

[3] There will be a test program macro processing facility.

[4] There will be a test program macro library.

[7] Incremental test program compilation will be supported when more test steps are added to the end of the test program.

[10] The GO/NOGO test program will be generated.

[11] There will be New-to-Current GO/NO-GO test program conversion facilities.

[12] A test program can contain multiple sets of input test patterns each testing a different part of the same board.

[13] A test program can contain multiple sets of input test patterns all of which test the same part of a board.

[14] There will be Current-to-New test program conversion facilities

SUBPROBLEM 2 -- CIRCUIT DESCRIPTION CODING AND PROCESSING

[5] Default driving characteristics associated with output or bidirectional pins for each standard device type will be supported.

[6] Default driving characteristics associated with output or bidirectional pins for each standard device type can be overriden by user.

APPENDIX C

[8] Simulation results for a device model can be overriden by user.

[9] Output of a device for which no simulation model exists can be controlled by user.

[15] There will be a Circuit Description Language which allows the user to specify board-related or IC-related descriptions.

[16] Format of Circuit Description Source file will be compatible with that of In-Circuit test systems.

[17] Different sections of the circuit descrition source file can be used by more than one product type if so specified.

[18] Different sections of the circuit description source can be shared by multiple sets of input patterns in the same test program which have the identical circuit topology.

[19] There will be a circuit description processing facility.

[20] There will be partial re-processing facility for circuit description source file.

[21] Default driving characteristics of output or bidirectional pins for each specific tester will be supported.

[22] Default driving characteristics of output or bidirectional pins for each specific tester can be overriden by user.

[23] Board-related statistical data will be reported to user.

[24] Board-related warnings will be reported to user.

[25] Circuit network connection cross-reference data will be reported to user.

[26] There will be Current-to-New circuit description source file conversion facilities.

APPENDIX C

[31] Default transport delay characteristics associated with output pins of each standard IC family type will be supported.

[34] IC's can be modeled using the circuit description technique.

[36] Default transport delay characteristics associated with output pins of each standard IC family type can be overriden by the user on a pin by pin base.

[43] There will be a conditional connectivity language for the user to define connectivity of an IC model.

[51] Default transport delay characteristics associated with output pins of each specific tester type will be supported.

[52] Default output pin transport delay characteristics for each tester type can be overriden by user.

[68] Default transport delay characteristics associated with output pins of each standard IC family type will be supported.

[69] Default transport delay characteristics associated with output pins of each standard IC family type can be overriden by user on a pin by pin base.

SUBPROBLEM 3 -- IC MODELS CONVERSION

[28] About 90% of the existing System IC model library will be converted.

[29] Every widely used primitive element and SSP Boot element will be supported.

[41] There will be tools aiding .SR-type IC models conversion.

[42] There will be tools aiding SCL-I TO SCL-II IC models conversion.

APPENDIX C

SUBPROBLEM 4 -- IC LIBRARY HANDLING

[27] There will be IC model libraries.

[30] Models for ROM's will be supported.

[32] There will be IC library maintenance facilities.

[33] There will be an IC model information display facility.

[35] Models for Gate Arrays (PLA devices) will be supported.

SUBPROBLEM 5 -- OUTPUT PATTERNS SIMULATION

[37] IC's can be modeled using the functional modeling technique (SCL-II).

[38] SCL-II will be convenient to use for both system manufacturer and customer.

[39] SCL-II will be efficient enough to be suitable for some in-house modeling of primitives.

[40] There will be SCL-II debugging tools.

[44] Connectivity of .SR-IC models can be automatically simulated using a connectivity simulator.

[45] There will be a (good board) simulator.

[47] The simulator will be as fast as possible.

[48] Results of good board simulation can be properly displayed.

[50] Output pin transport delay characteristics will be taken into account by the simulator.

[53] Bus type logic will be properly handled.

[54] Properties of a particular target tester will be taken into account by the simulator.

APPENDIX C

ıe tester will be better synchronized to the
ımulator.

ıternal node faulty activity for an undetected fault
ın be displayed to user.

SUBPROBLEM 6 -- FAULT MODELING AND INSERTION

ıe Stuck type fault models will be supported.

ıere can be user inserted faults.

ıer inserted faults will be consistent with fault
ısertion machinary.

ıme form of LSI internal fault models will be
ıpported.

SUBPROBLEM 7 --
'ROGRAM EVALUATION AND FAULT ISOLATION PREPARATION

ıere will be fault simulation activity.

ıcremental simulation will be supported when more
:eps are added at the end of the test program.

ıult simulation will be as fast as possible.

ıst program fault coverage scoring will be reported
ı user.

ıtected/Undetected faults will be reported to user.

:OBLEM 8 -- TEST PROGRAM MACRO PROCESSING FACILITY

'here will be a test program macro processing
'acility.

'here will be a test program macro library.